Uma abordagem da automação distribuída com sistemas embutidos utilizando tecnologias multiagentes para resolução do *puzzle*-8

Fabiano Tomás Novais, Frederico Gadelha Guimarães
PPGCC - Programa de Pós-Graduação em Ciência da Computação
UFOP - Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
email: fabianotomasnovais@gmail.com, frederico.g.guimaraes@gmail.com

***Resumo*—Neste trabalho desenvolvemos e avaliamos dois métodos de resolução do problema do *puzzle*-8, um utilizando uma abordagem Multiagente e outro com algoritmo de Tentativa e Erro. O problema do puzzle-8 foi utilizado devido ao fato dele ser amplamente utilizado, pela sua simplicidade e por ser um problema NP-difícil. Foi utilizado também uma abordagem distribuída, visando disponibilizar as soluções obtidas a sistemas embutidos, os quais apresentam recursos limitados para resolução de problemas como o *puzzle*-8.**

***Keywords*-multiagente, sistemas embutidos, automação distribuída.**

I. Introdução

A automação, antes restrita as indústrias, hoje esta cada vez mais presente em nossas vidas, desde um microondas, quando programamos o aquecimento de alimentos, até nos nossos carros, como nos sistemas com injeção eletrônica. A automação pode ser empregada no aumento da segurança, na redução de gastos, para comodidade das pessoas, no aumento da eficiência de um processo, no aumento da produção, *etc*.

Os sistemas embutidos são dispositivos dotados de certa inteligência computacional, capazes de processar informações ou aprender conhecimento por meio de técnicas especiais, porém dedicados ao desenvolvimento de uma única atividade ou um grupo limitado de atividades. Estes dispositivos frequentemente são utilizados para tarefas de monitoramento e controle de equipamentos, tais como sensores, motores e chaves. Eles recebem o nome "embutido", pelo fato do programa de controle estar gravado junto ao circuito que o executa [1].

Apesar de terem vários recursos de *hardware*, como conversores analógicos, interface para comunicação, memória interna, os sistemas embutidos ainda são dispositivos limitados, principalmente no que se refere a capacidade de processamento. Muitas das vezes tais sistemas necessitam de uma maior capacidade de processamento, o que os torna limitados para certas aplicações. Uma maneira de aumentar sua capacidade é utilizar sistemas distribuídos de forma que servidores distribuídos com grande capacidade de processamento possam dispor dos seus recursos para o processamento de dados enviados pelo sistema embutido por meio de uma rede de comunicação como a internet.

Segundo [2] um agente é qualquer coisa que pode perceber o ambiente por meio de sensores e atuar sobre este ambiente por meio de atuadores. Um agente humano tem olhos, ouvidos, e outros órgãos como sensores, pernas, boca, e outras partes do corpo como atuadores. Em geral supomos que todos agentes podem perceber suas próprias ações (mas nem sempre os efeitos).

Conforme [3] um motivo essencial para a aplicação de tecnologias de agentes tem sido a possibilidade de utilizar métodos coordenados e distribuídos de agentes. Problemas que podem ser modelados naturalmente como sistemas agentes tem sido as principais aplicações. Estes problemas podem tipicamente ser decompostos corretamente em subproblemas que podem ser solucionados de forma quase independente.

Entretanto, aplicações com tecnologia de agentes em processos de automação não tem sido numerosas. Uma razão para isto seria provavelmente as exigências de execução em tempo real nas aplicações em processos de automação que atualmente as tecnologias de agentes dificilmente podem executar. Uma outra razão seria a característica das tarefas de controle de processo apresentarem um complexo inter-relacionamento entre várias variáveis de controle. Isto tornaria difícil a tarefa de encontrar decomposições dos problemas que são adequados para os agentes. Uma terceira razão poderia ser a raridade de paralelismo e de recursos redundantes em processos controlados, que muitas vezes são modelados como agentes em aplicações de outras áreas como a de controle da produção.

De acordo com [4] o objetivo da pesquisa em Sistemas Multiagentes é encontrar métodos que nos permitam construir sistemas complexos compostos por agentes autônomos que, operando com conhecimentos locais e que possuindo apenas capacidades limitadas, são, todavia, capazes de encenar os comportamentos globais desejados. Sistemas Multiagentes abordam problemas usando as ferramentas comprovadas de teoria dos jogos, Economia e Biologia. Ela complementa isto com conceitos e algoritmos de pesquisa de inteligência artificial, ou seja, planejamento, métodos de raciocínio, os métodos de pesquisa e aprendizado de máquina.

Conforme [5] o *puzzle*-8 é o maior problema do seu tipo

que pode ser completamente resolvido. Ele é simples, e ainda obedece a um grande problema combinatório com espaço de $9!/2$ estados possíveis. A extensão de N x N do *puzzle*-8 é NP-difícil. O objetivo do *puzzle*-8 é reorganizar uma configuração inicial de oito azulejos quadrados em um tabuleiro 3x3 em uma configuração especifica por meio de sucessivos deslizamentos dos azulejos que são ortogonalmente adjacentes ao vazio (o quadrado em branco).

O artigo está organizado da seguinte forma. A Seção II apresenta a metologia adotada no trabalho. A Seção III apresenta a solução do *puzzle*-N com uma abordagem Multiagente. A Seção IV apresenta a solução do *puzzle*-N com um algoritmo de Tentativa e Erro. Na Seção V é apresentado o sistema embutido proposto para automação distribuída. A seção VI apresenta a análise de complexidade dos métodos adotados. Os experimentos podem ser vistos na Seção VII. E por fim a conclusão na Seção VIII.

II. Métodos

Inicialmente foi realizado o levantamento bibliográfico sobre os Sistemas Multiagentes (SMA) e das principais aplicações de SMA em sistemas embutidos e automação distribuída.

A fim de realizar a comunicação dos sistemas embutidos com os servidores, foi adotado a rede Internet devido está já estar bem difundida e ser compatível com a tecnologia Multiagentes.

Os sistemas embutidos foram dotados de uma interface *ethernet* e recursos como sensores de temperatura e corrente, nesse caso cada dispositivo foi considerado um agente assim como os servidores que rodam os dois métodos de resolução do *puzzle*-8.

Como forma de avaliar o desempenho na solução de problemas, o sistema desenvolvido foi exposto ao problema do *puzzle*-8 que é de difícil resolução para sistemas embutidos. Como forma de resolver este problema, duas abordagens foram desenvolvidas, sendo uma utilizando um algoritmo de Tentativa e Erro, e o outro utilizando Sistemas Multiagentes.

III. Solução por meio de Sistemas Multiagentes

A. Decompondo do Problema

De forma a resolver um *puzzle*-8 ou um *puzzle*-N, utilizou-se a seguinte metologia de acordo com [6]. Utilizando uma abordagem distribuída, inicialmente decompomos o problema de resolver um *puzzle*-N em subproblemas, onde cada uma das n peças do tabuleiro do *puzzle* tenta alcançar um objetivo específico. O objetivo geral consiste em satisfazer todos os subproblemas. Assim, se G é a meta global, temos:

$$G = \{g_1, ..., g_n\} \quad (1)$$

onde $\{g_1, ..., g_n\}$ são as metas independentes que precisam ser alcançadas para G ficar satisfeito. Para o *puzzle*-N isto é, cada peça alcançar o caminho em um determinado tempo. Podemos então declarar:

$$G_{puzzle} = \{posicao(t_1, p_1), ..., posicao(t_1, p_1)\} \quad (2)$$

onde t_1 e p_1 são respectivamente as peças e suas posições objetivo.

Descrevemos a satisfação de cada objetivo com uma função Booleana de um conjunto de dois elementos que chamamos de agentes:

$$\forall_{g_i} \forall_{a_i} = g_i, satisfeito(g_i) = f(a_i, meta(a_i)) \rightarrow 0, 1 \quad (3)$$

onde a_i é uma peça ou o componente individual do problema e $meta(a_i)$ é o objetivo (outro componente) a ser alcançado.

A submeta g_i vai ser considerada satisfeita se e somente se $f(a_i, meta(a_i))$ retorna verdadeiro.

$$satisfeito(posicao(t_1, p_1)) = sobre(t_1, p_1)) \quad (4)$$

Mudando o ponto de vista e considerando agora o agente a_i, isto é a mesma coisa de dizer que g_i está satisfeito se e somente se a_i alcança sua meta. Podemos descrever a_i com a seguinte tríplice:

$$\forall_{g_i}, a_i =< meta(a_i), estado(a_i), comp(a_i) > \quad (5)$$

onde a_i é o mesmo que o visto na equação 3, $estado(a_i)$ é o estado corrente do agente, e $comp(a_i)$ um conjunto de intenções de ações para alcançar o objetivo do estado corrente. Este estado atual é definido, a fim de permitir a seguinte definição:

$$\forall_{g_i} \forall_{a_i} / meta(a_i) = g_i, satifaca(g_i) \Leftrightarrow estado(a_i) = meta(a_i) \quad (6)$$

Se aplicarmos estas definições para o *puzzle*-N, podemos descrever as peças t_i da seguinte maneira:

$$t_i = \{p_i, p_k, comp(t_i)\} \quad (7)$$

$$satisfaca(posicao(t_i, p_i)) \Leftrightarrow p_k = p_i \quad (8)$$

onde p_i é a meta de t_i, p_k é a posição na qual t_i está localizado e $comp(t_i)$ um conjunto de ações que permitem sair da posição corrente para alcançar sua meta. Com isso basta que cada peça faça "a coisa certa" para alcançar sua submeta de forma que após todas submetas serem satisfeitas a meta global também estará.

Um problema P é descrito agora como uma coleção de agentes, seus estados iniciais $Inicio_P$ dados pelos seus estados correntes e seus comportamentos e seus estados finais $Final_P$, como declarados a seguir:

$$P = \{a_1, ..., a_n\} \tag{9}$$

$$Inicio_P = \{estado(a_1), comp(t_1), ..., estado(a_n), comp(t_n)\} \tag{10}$$

$$Final_P = \{meta(a_1), ..., meta(a_n)\} \tag{11}$$

O problema é então considerado como solucionado quando todos agentes tenham alcançado suas metas, o que significa que todas as submetas tenham sido alcançadas. Instanciando estas definições para o *puzzle-N*, obtemos:

$$Puzzle = \{t_1, ..., t_n, p_1, ..., p_n\} \tag{12}$$

$$Inicio_{Puzzle} = \{(p\alpha), comp(t_1), ..., (p\lambda), comp(t_1)\} \tag{13}$$

$$Final_{Puzzle} = \{meta(t_1) = p_i, meta(t_n) = p_j\} \tag{14}$$

B. Modelo Eco-Problem-Solving

De acordo com [6], este modelo é baseado no paradigma de "computational eco-systems". Neste modelo a resolução de problemas é vista como a produção de estados estáveis em um sistema dinâmico, onde a evolução é devido ao comportamento dos agentes simples. Por "simples", queremos dizer que os agentes não fazem planos, mas apenas comportam de acordo com seu programa específico, chamado de eco-comportamento, que é apenas uma combinação de "tropismos", ou seja, comportamentos feitos de reações ao ambiente.

Um modelo *Eco-Problem-Solving* é construído a partir de duas partes: **(1)** um núcleo independente onde eco-comportamentos são descritos. Isto consiste de uma definição resumida dos eco-agentes e de seus protocolos de interação. **(2)** Uma divisão da aplicação onde as ações invocadas por estes comportamentos são codificadas. O núcleo dos eco-comportamentos são caracterizados pela:

- A vontade de ser satisfeita: Isto corresponde à descrição da meta. Uma função chamada *tentaSatisfazer* cuida disto. Esta função chama duas ações : *satisfacaSemAtacar* (se o agente pode ser satisfeito movendo para o vazio) ou *satisfacaComAtaque* (se ele precisa atacar outros agentes para conseguir satisfazer-se). Um agente satisfeito não busca satisfazer-se mais, a menos que seja dado a ele uma nova meta.
- Tentativa de Fugir: É a resposta a um ataque. Isto corresponde ao agente sair de sua posição atual no problema para evitar conflitos. A função *tentaFugir* faz isto e leva a duas ações: *fugaSemAtaque* (se existe uma maneira de fugir) ou *fugaComAtaque* (se ele deve atacar outros agentes). Uma mensagem de *fugir* é fornecida com uma restrição (o próprio atacante) pelo *atacante* a outro agente. O agente em fuga, então, não terá a possibilidade de atacar esta restrição a menos que ele não tenha outra opção. Neste caso temos um contra ataque, o que faz a tentativa de ataque do *atacante* fracassar além de levá-lo a fugir do contra-ataque.

Podemos então definir mais precisamente os agentes como:

$$\forall_{a_i}, a_i =< meta(a_i), estado(a_i), comp(a_i), fugir(a_i) > \tag{15}$$

É importante notar que os agentes somente atuam quando estão tentando ser satisfeitos ou quando estão tentando fugir de um ataque para a posição mais próxima do vazio. Além disso estas decisões são tomadas sem o conhecimento global do estado do problema, apenas levando em conta o conhecimento local.

C. Ordenando as Submetas

Após decompor o problema do *puzzle-N* em subproblemas devemos ordenar a ação de cada agente, devido ao fato de não podermos mover mais de uma peça ao mesmo tempo, por haver somente um espaço vazio. Além disso devemos adotar um sequência lógica que nos leve a solução.

A ordem adotada parti das linhas e colunas mais distantes do vazio percorrendo todos com a mesma distância do vazio, para só depois percorrer os com menor distância. Este caminho se assemelha a um redemoinho e sua escolha foi motivada pelo fato que após as linhas e colunas mais externas estiverem devidamente posicionadas, o restante pode ser solucionado sem perturbar-las.

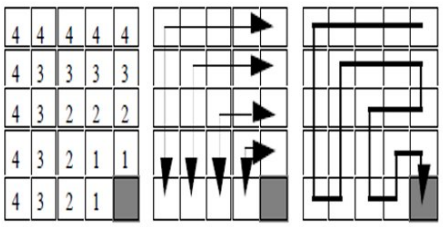

Figura 1. Caminho: Ordem usada para satisfazer os agentes

Na figura 1 podemos ver o caminho e as distâncias de cada peça ao vazio. Este caminho é obtido atribuindo a cada peça ou agente, a distância de *Manhattan* das linhas ou colunas a que pertencem até o vazio. Sendo distância de *Manhattan* para duas peças nas posições $p_1(x_1, y_1)$ e $p_2(x_2, y_2)$ dada pela seguinte equação 16. A distância de *Manhattan* também é utilizada para encontrar as posições objetivo nas tentativas de satisfazer um agente e nas tentativas de fuga.

$$distancia(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2| \qquad (16)$$

D. Conhecimento dos agentes

Cada agente é composto por um conjunto de componentes necessários para sua interação com o meio em que ele está em contato. No caso do *puzzle*-N, o ambiente é o tabuleiro e as outras peças. Sendo assim torna-se necessário o agente ter algum conhecimento do ambiente em que ele está, para poder ter certos comportamentos de acordo com o meio e a situação atual.

Os principais conhecimentos do agente são:

- $Conh_0$: Localização do tabuleiro (onde o tabuleiro é aqui representando por uma matriz nxn de agentes);
- $Conh_1$: Estado atual (no caminho, bloqueado, atacando, fugindo, aguardando);
- $Conh_2$: Se está satisfeito;
- $Conh_3$: Qual agente está atacando;
- $Conh_4$: Qual número das peça do tabuleiro ele está representando;
- $Conh_5$: Onde está o vazio (o vazio é o agente a_{n-1} com o número 0);
- $Conh_6$: Meta atual a_i;
- $Conh_7$: Posição atual no tabuleiro $p(a_i)$.

E. Comportamentos e restrições dos agentes

A seguir listamos os principais comportamentos adotados para cada agente (ou posição) na definição de qual caminho seguir de acordo com suas restrições e as do agente selecionado como nova meta:

- $Comp_0$: O agente escolhido não é o próprio agente que o atacou e não está fugindo e nem atacando e não esta em um caminho bloqueado, então aceita como meta;
- $Comp_1$: Se o agente estiver satisfeito (bloqueado para sair do caminho), só troca de posição se for para o mesmo caminho;
- $Comp_2$: Se a posição mais próxima for a posição objetivo, então aceita como meta;
- $Comp_3$: Se encontrou uma distância menor em relação a atual ou que esteja em melhor estado, então aceita como meta;
- $Comp_4$: Caso a posição mais próxima seja o vazio, aceita como meta;
- $Comp_5$: Caso tenha encontrado uma posição com mesma distância mas com agente não bloqueado nem atacando, aceita como meta;
- $Comp_6$: Só permite ataque de agentes não bloqueados a agentes bloqueados, o seja num caminho já formado, caso o agente selecionado seja o último do caminho.

F. Resolvendo o puzzle-N

O fluxograma da figura 2 mostra os principais passos para encontrar a solução do *puzzle*-N.

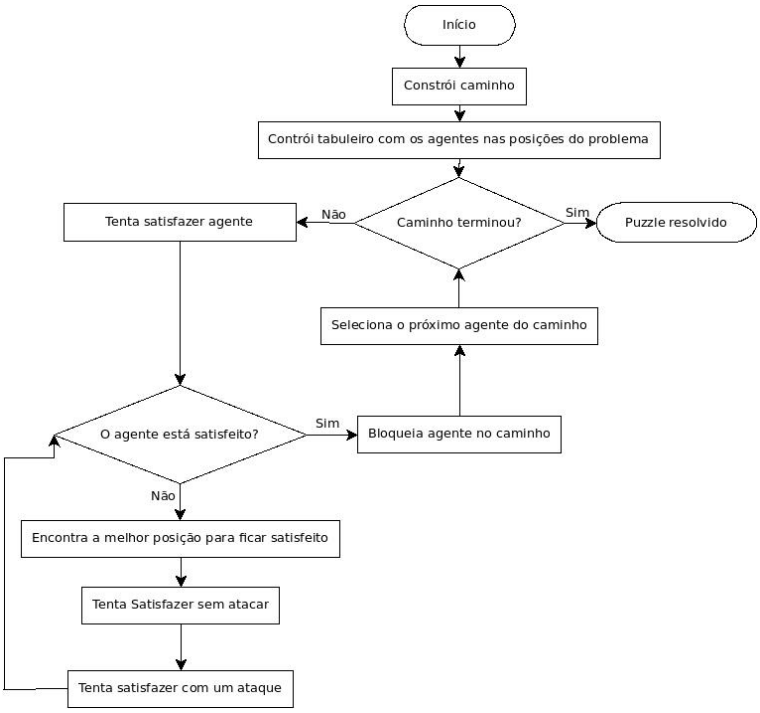

Figura 2. Fluxograma do programa usando abordagem multiagente

A figura 3, a qual utiliza a abordagem de [6], não é utilizado o contra-ataque de mais de uma peça na tentativa de satisfazer o agente A, porém os efeitos são os mesmos, havendo apenas uma diferença da utilizada neste trabalho na qual as peças B e C se movem juntamente com E num contra-ataque a A.

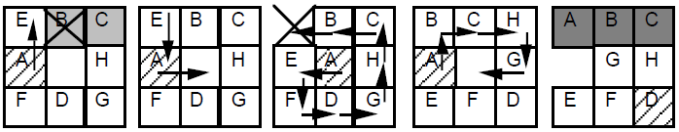

Figura 3. Agente A tentando ser satisfeito

IV. Solução com Algoritmo de Tentativa e Erro

A. Algoritmo de Tentativa e Erro

De acordo com [7] tentativa e erro é o processo de decompor um número finito de sub-tarefas parciais e explora-las exaustivamente.

Suas principais características são:

- O processo de tentativa gradualmente constrói e percorre uma árvore de sub-tarefas;
- Algoritmos tentativa e erro não seguem uma regra fixa de computação;
- Passos em direção à solução final são tentados e registrados;

- Caso esses passos tomados não levem à solução final, eles podem ser retirados e apagados do registro;
- Quando a pesquisa na árvore de soluções cresce rapidamente é necessário usar algoritmos aproximados ou heurísticas que não garantem a solução ótima mas são rápidas.

B. Pseudo Código

```
//Tenta resolver dada a posicao atual do vazio
int tentaPuzzle(int x,int y)
{
//Inicializa tentativas
tentpossivel = 0
//Armazena as posicões anteriores
salva(x,y)
//Armazena a posicao atual do vazio
salvaAtual()
//Incrementa a profundidade
profundidade++
//Verifica se o tabuleiro esta ordenado
ordenado = tabuleiroOrdenado()

Enquanto nao encontra uma solucao executa
    while(tentpossivel<4 && !solucao())
    {
//Seleciona uma nova posicao das tentativas
    possíveis
posicao = seleciona()
//Verifica se ela e valida e senao e uma posicao
    anterior a atual e se a profundidade e menor
    do que 31
        if(verfica(posicao) && profundidade<p)
        {
//Troca elementos de lugar no tabuleiro de lugar
trocaPosicao(posicao)
//Insere quais posicões devem ser trocadas no
    conjunto solucao
            insereTroca()
//Enquanto nao encontra uma solucao executa
            if(!solucao())
            {
//Tenta puzzle para nova posicao se a posicao
    atual nao corresponder a primeira posicao do
    tabuleiro
                posicao = posicaoInicial()
                if(posicao==0)
                    tentaPuzzle(xvazio,yvazio)

//Verifica se conseguiu ordenar
                    if(!solucao)
                    {
//Retorna os elementos para o lugar original
            retornaPosicoes()
//Remove posicao do conjunto solucao
                        removeTroca()
                    }
                    else
                        break
            }
            else
                break
        }
        tentpossivel++;
//Retorna os elementos para o lugar original caso
    tenhan tentado todas possibilidades
if(tentpossivel==4)
  retornaPosicoes()

  }
```

V. Sistemas Embutidos atuando como agentes

O sistema embutido utilizado no projeto pode ser visto na figura 4, ele é composto de um *display* de LCD de 128x64 *pixels*, uma tela *touch screen*, uma interface *ethernet* para comunicação, um microcontrolador Pic18f4620 e de outros periféricos.

Como pode ser visto este sistema não é o ideal para resolver um problema como *puzzle-N* devido as limitações de *hardware*. Porém utilizando uma abordagem distribuída, na qual cada sistema embutido atua como um agente, podemos por meio de outros agentes encontrar a solução para o problema.

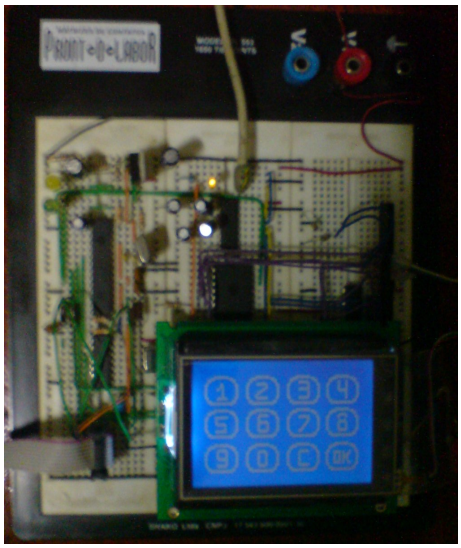

Figura 4. Sistema Embutido

VI. Análise de Complexidade

A seguir será realizado a análise de complexidade dos dois algoritmos propostos quanto a complexidade de tempo e espaço.

A. Sistema Multiagentes

A complexidade de espaço do algoritmo proposto para resolução do *puzzle-N* utilizando Sistemas Multiagentes é proporcional ao número de agentes, pois o algoritmo não mantém os estados dos problemas na memória. Sendo assim a complexidade de espaço do algoritmo é:

$$SP(N) = O(N) \tag{17}$$

Para a complexidade de tempo supomos que cada agente saia da posição mais a esquerda e da parte inferior até a parte superior e mais a direita ou vice e versa, obtemos então $2n-1$ movimentos, supondo também que após bloqueado

ele realize 4 passos, devido a inserção das últimas peças do canto superior e do canto inferior, temos então um total de $2n + 3$ movimentos.

Logo para o caminho com distância n do vazio, temos os custos para cada agente pertencente a este caminho:

$$a_n = (2n + 3)(2n - 2) = (4n^2 + 2n - 6) \quad (18)$$

Assim como existem $n - 1$ caminhos com distâncias diferentes do vazio, obtemos:

$$T(n) = \sum_{1}^{n} = (4n^2 + 2n - 6) = (9n^3 + 9n^2 - 13n + 18)/3 \quad (19)$$

Logo o algoritmo é $O(n^3)$.

B. Tentativa e Erro

Com no algoritmo de Tentativa e Erro apresentado para resolução do *puzzle*-N existe um limite de p chamadas recursivas (limite de profundidade $p = 31$ para o *puzzle*-8), temos que a complexidade de espaço do algoritmo é:

$$SP(N) = O(p) = O(1) \quad (20)$$

Este limite p corresponde ao número mínimo de movimentos para resolver um *puzzle-N*.

Considerando que o espaço de busca de um puzzle N apresenta $N!$ estados e que apenas metade deste espaço é alcançável, a complexidade de tempo no pior caso é:

$$SP(N) = O(N!/2) = O(N!) \quad (21)$$

VII. Experimentos

Nos experimentos foram avaliados o número de trocas e o tempo para a resolução do problema de um *puzzle*-8, conforme pode ser visto nas Tabelas I e II. Pode se notar que o tempo de execução dos algoritmos variam muito de acordo com o problema a ser resolvido, porém o número de trocas do algoritmo com Sistemas Multiagentes é muito menor que o de Tentativa e Erro. Mesmo assim em algumas situações o de Tentativa e Erro respondeu mais rápido, o que o torna uma alternativa num ambiente com vários agentes em que o tempo é crucial. A média de movimentos para alcançar a solução de um problema do *puzzle*-8 é 36 para o algoritmo com sistemas Multiagentes e 28 para o de Tentativa e erro.

Tabela I
TEMPO DE EXECUÇÃO

Tempo de execução em milissegundos	
Sistema Multiagente	Algoritmo de Tentativa e Erro
23	24
23	20
12	19
69	22
76	80
55	30
89	22
26	22

Tabela II
NÚMERO DE TROCAS DE PEÇAS

Número de trocas	
Sistema Multiagente	Algoritmo de Tentativa e Erro
29	1708386
24	270700
28	54664
29	338115
26	31407
10	140000
18	45631
21	156000

VIII. Conclusão

Neste trabalho foram apresentados duas abordagens para resolução do *puzzle*-N, uma utilizando Sistemas Multiagentes e outra com um algoritmo de Tentativa e Erro. Como pode ser observado a abordagem com Sistemas Multiagentes possibilita resolver problemas grandes em tempo polinomial, porém as soluções encontradas não são ótimas. Já o algoritmo o algoritmo de Tentativa e Erro para o puzzle-8, conseguiu encontrar a solução, porém em um tempo maior e sem garantias dela ser ótima.

Com os resultados obtidos das duas abordagens, cabe agora o sistema embutido (ou agente embarcado) requisitar a resolução do problema e selecionar a melhor ou a que for respondida em menor tempo. Assim fica como trabalho futuro a criação de um protocolo de comunicação entre os sistemas embutidos que permita a troca de informação para resolução de problemas complexos como o *puzzle*-N.

Referências

[1] Vago, "O uso da internet e da comunicacao sem fio via zigbee em sistemas embutidos," Universidade Federal de Ouro Preto, Ouro Preto, Technical Report, 2008.

[2] S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, 2nd ed. Prentice Hall, 2003, ISBN-13: 790395-2.

[3] I. Seilonen, P. Appelqvist, M. Vainio, A. Halme, and K. Koskinen, "A concept of an agent-augmented process automation system," in *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, 2002, pp. 473 – 478.

[4] J. Vidal, "Fundamentals of multiagent systems," 2007.

[5] A. Reinefeld, "Complete solution of the eight-puzzle and the benefit of node-ordering in ida*," 1993, pp. 248–253.

[6] A. Drogoul and C. Dubreuil, "A distributed approach to n-puzzle solving," in *Proceedings of the Distributed Artificial Intelligence Workshop*. Citeseer, 1993.

[7] L. A. A. Ferreira, "Paradigmas de projeto de algoritmos," http://homepages.dcc.ufmg.br/~loureiro/alg/111/paa_01AnaliseDeComplexidade.pdf,.