

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DANIELA DOS SANTOS
DENISE DE OLIVEIRA
FABIANA LORENZI
PAULO ROBERTO FERREIRA JR.

**Estudo de Insetos Sociais e sua aplicação na
solução de problemas computacionais**

Relatório de Pesquisa
RP-350

Prof^a. Dra. Ana Lucia Cetertich Bazzan
Orientador

Porto Alegre, Janeiro de 2005

SUMÁRIO

RESUMO	3
1 INTRODUÇÃO	4
2 ALOCAÇÃO DE TAREFAS	5
2.1 Dança das Abelhas	6
2.2 Aplicação: Sistemas de Recomendação	8
3 SATISFAÇÃO DE RESTRIÇÕES	10
3.1 Formalização do CSP	11
3.2 <i>Ant-Solver</i>	12
3.3 Satisfação de Restrições Distribuídas	13
4 CONCLUSÃO	15
REFERÊNCIAS	16
APÊNDICE A	18
APÊNDICE B	29

RESUMO

Este relatório refere-se ao conteúdo visto na disciplina CMP301 e contém um estudo sobre alocação de tarefas e satisfação de restrições em colônias de insetos sociais. Ainda no tópico de alocação de tarefas, mostra-se um estudo realizado sobre a abordagem da Dança das Abelhas, bem como sua utilização no desenvolvimento de um sistema de recomendação. Por fim, são apresentados os resultados dos experimentos realizados neste trabalho.

Palavras-chave: Inteligência Artificial, *Swarm Intelligence*, Insetos Sociais, Raciocínio Baseado em Casos, Sistemas de Recomendação, Satisfação/Otimização de Restrições.

1 INTRODUÇÃO

As colônias de insetos sociais - formigas, abelhas, cupins e vespas - são sistemas descentralizados que possuem características desejáveis para a solução de diversos problemas computacionais. Pesquisas anteriores sobre a organização das colônias de insetos sociais e sua aplicação na organização de Sistemas Multiagentes (MAS) mostraram bons resultados. A proposta desta disciplina foi aprofundar o estudo desta abordagem e sua aplicação em Sistemas de Recomendação para a web, em novos problemas de organização em MAS e em problemas de satisfação de restrições.

Este relatório está dividido em dois tópicos estudados durante o decorrer da disciplina: o capítulo 1 descreve como funciona a alocação de tarefas nas colônias de insetos sociais e especificamente a abordagem da dança das abelhas. O capítulo 2 descreve o tópico de Satisfação de Restrições.

2 ALOCAÇÃO DE TAREFAS

Colônias de Insetos Sociais são grupos de indivíduos que vivem juntos e se reproduzem como uma unidade. A colônia representa um nível de organização acima dos organismos individuais com suas próprias características morfológicas, comportamento, organização interna e padrão de desenvolvimento. Os biólogos que estudam os insetos sociais enfrentam o desafio de entender o comportamento da colônia a partir do entendimento do comportamento de um único indivíduo.

Os modelos mais conhecidos de organização focam na adaptação da estrutura social das colônias. Os modelos mais recentes tratam as colônias como um sistema descentralizado e auto-organizado cujo comportamento emerge da ação e decisão dos indivíduos. Modelos auto-organizados têm sido usados para descrever inúmeros processos dentro das colônias.

A divisão de trabalho, onde indivíduos diferentes se especializam em um subconjunto das tarefas realizadas na colônia, é uma das características mais impressionantes do comportamento de uma colônia. Beshers e Fewell (BESHERS; FEWELL, 2001) discutem os modelos atuais de divisão de trabalho, analisando de forma crítica suas características e resumindo o estado da arte nesta área. Neste artigo são descritas seis classes de modelos, baseadas nas causas da divisão do trabalho: limiar de resposta, transferência de informação, auto-reforço, inibição social, forrageando por trabalho e modelos de alocação de tarefas em rede.

Os modelos baseados em limiar de resposta são baseados na hipótese que os indivíduos têm limiares internos para responder ao estímulo produzido por cada tarefa e que a variação deste limiar entre os indivíduos gera a divisão de trabalho. Esta variação pode se dar através de uma distribuição inicial diferenciada, caracterizando diferenças no genótipo dos indivíduos, ou através da mudança dos limiares durante a realização das tarefas, caracterizando um sistema de especialização em tarefas. Os modelos de auto-reforço e de transferência de informação são integrados ao modelo de limiar de resposta. No modelo de troca de informação, os estímulos utilizados no modelo de limiar de resposta provém das tarefas diretamente e da troca de informação entre os indivíduos, que são capazes de recrutar outros indivíduos para a realização de uma tarefa. O modelo de auto-reforço se baseia na especialização dos indivíduos na realização de determinadas tarefas. O limiar de resposta do indivíduo diminui para a tarefa que ele está realizando a medida que ele a realiza.

Os modelos baseados no forragear pro trabalho baseiam-se na busca aleatória por tarefas que estejam demandando trabalho. Essencialmente, estes modelos são baseados em um algoritmo simples: (a) trabalhe em qualquer tarefa que esteja demandando trabalho; (b) uma vez que a tarefa está sendo realizado, continue trabalhando na mesma tarefa; (c) se a tarefa não demanda mais trabalho, mova-se para um outro lugar e procure por uma

nova tarefa com demanda. Em suma, nestes modelos a divisão emerge da distribuição espacial das tarefas.

Os modelos de inibição social são baseados na interação de um processo de desenvolvimento comportamental intrínseco ao indivíduo inibido pela interação social com outros indivíduos. Isto significa que o estado fisiológico do indivíduo que varia com seu amadurecimento pode ser desacelerado ou até revertido pela interação social. Estes modelos, conhecidos inicialmente como "inibidores-ativadores", atrelavam o processo de desenvolvimento do indivíduo somente às quantidades de hormônio da juventude em seu organismo, o que atualmente é considerado apenas mais um fator que atua nesse processo.

Finalmente, os modelos de alocação de tarefas em rede são baseados na interação simples entre os indivíduos. Estes modelos assumem que não existem diferenças intrínsecas entre os indivíduos e que a distribuição do trabalho ocorre pela comunicação entre os indivíduos. Através dessa comunicação, os indivíduos divulgam entre si a quantidade de trabalhadores atuando em cada tarefa.

No artigo (BESHERS; FEWELL, 2001), os autores exploram as evidências empíricas de cada modelo, destacando que provavelmente a natureza se vale de todos os mecanismos descritos pelo modelos discutidos anteriormente para obter um sistema tão sofisticado e eficiente de divisão de trabalho.

A seguir será discutido um dos modelos baseados na transferência de informações.

2.1 Dança das Abelhas

A pesquisa realizada por Camazine et al (CAMAZINE et al., 2003) descreve a chamada Dança das Abelhas. Esta técnica também é chamada de dança de recrutamento, executada por uma abelha quando uma fonte de néctar muito rica foi visitada.

A atividade forrageadora começa a cada manhã quando centenas de abelhas voam em busca de fontes de néctar (CAMAZINE et al., 2003). Uma abelha pode também chegar até uma fonte de néctar seguindo a dança de uma companheira que já descobriu tal fonte. Dentro da colméia existe uma área denominada "pista de dança", a qual contém abelhas dançando para diferentes fontes de néctar. A dança transmite informações a respeito da fonte, ou seja, distância, direção e qualidade. As abelhas que estão na pista de dança observando as dançarinas, selecionam uma das colegas e voam para a fonte correspondente a fim de coletar seu néctar. Imediatamente após a coleta, a abelha retorna para a colméia e decide se:

- Abandona a fonte e se dirige para a pista de dança para observar outra colega dançarina;
- Realiza dança de recrutamento antes de voltar a forragear em sua fonte de néctar;
- Simplesmente continua forrageando sem recrutar companheiras

Os fatores que influenciam a tomada de decisão descrita anteriormente são: a qualidade da fonte de néctar, distância, direção e facilidade de acesso.

Em (SEELEY; CAMAZINE; SNEYD, 1991), os autores apresentaram um experimento mostrando o recrutamento de abelhas para fontes contendo solução de açúcar. Duas fontes foram apresentadas para a colônia às 8 horas: fonte A com uma concentração de 1.00mol de açúcar e fonte B com uma concentração de 2.5mol. Ao meio dia, as fontes foram trocadas: fonte A agora contém 2.5mol e fonte B 0.75mol. Nas duas situações a

melhor fonte foi mais visitada. O experimento mostrou que a decisão de forrageamento de cada abelha é baseada em limitadas informações sobre sua particular fonte visitada. Este comportamento simples permite que a colônia sempre selecione a melhor fonte de néctar para forragear.

Camazine e Sneyd desenvolveram um modelo matemático que demonstra como as propriedades do sistema emergem automaticamente das interações dinâmicas entre os componentes constituintes.

Figura 2.1: Modelo matemático de Sneyd (CAMAZINE; SNEYD, 1991)

A figura 2.1 mostra a estrutura do modelo que consiste de duas fontes de néctar e uma colméia de abelhas. Estas podem se encontrar em um dos sete compartimentos descritos a seguir:

- H_a : descarregando néctar da fonte A;
- H_b : descarregando néctar da fonte B;
- D_a : dançando para a fonte A;
- D_b : dançando para a fonte B;
- A: fonte A;
- B: fonte B;
- F: seguindo uma abelha dançarina.

Dois fatores afetam a proporção de abelhas em cada um dos sete compartimentos: a taxa na qual as abelhas movem-se de um compartimento para outro r_{1-7} , e a probabilidade de uma abelha seguir um ou outro caminho, pontos pretos na figura 2.1.

Após a abelha descarregar seu néctar na colméia, ela encontra o seu primeiro ponto de decisão: abandonar a fonte B (P_X^B) ou continuar na fonte B ($1 - P_X^B$). Se escolher continuar na fonte, então poderá decidir também se recruta novas abelhas para aquela fonte ($P_d^B(1 - P_X^B)$) ou não. Se a abelha decidir abandonar a fonte ela pode assistir as colegas dançarinas e decidir qual delas irá seguir (P_F^B). Esta probabilidade é estimada por:

$$B(P_F^B) = \frac{D_B d_B}{D_B d_B + D_A d_A}$$

Onde:

- d_A e d_B indicam a proporção de tempo que forrageadoras dançam, pois somente uma porção do tempo na área de dança é gasto dançando;
- D_A e D_B indicam a quantidade de abelhas na fonte A e B respectivamente.

A seguir descreve-se como a metáfora do comportamento das abelhas é aplicada no desenvolvimento de sistemas de recomendação.

2.2 Aplicação: Sistemas de Recomendação

Os sistemas de recomendação atuam na web fornecendo sugestões ao usuário de acordo com o contexto em que este se encontra. Estas recomendações geralmente são geradas a partir de um perfil adquirido pelo sistema de forma explícita, feedback, ou implícita, histórico do usuário, ou ainda utilizando-se das técnicas de Raciocínio Baseado em Casos.

Raciocínio Baseado em Casos (RBC) é uma metodologia utilizada para solucionar problemas por meio da reutilização de casos anteriores já conhecidos. Em um Sistema de Recomendação Baseado em Casos (RBC-RS) um conjunto de sugestões de produtos é recuperado da base de casos na busca de um caso similar ao caso descrito pelo usuário (BURKE, 2000).

O processo de recomendação respeita as seguintes etapas:

1. O usuário descreve algumas características do produto desejado;
2. O sistema busca na base de casos produtos similares as necessidades do usuário;
3. Um conjunto de casos é recuperado da base de casos e recomendado para o usuário;
4. Se o usuário não ficar satisfeito com a recuperação, poderá refinar sua requisição e um novo ciclo de recomendação é iniciado.

O processo de recuperação do caso pode ser considerado o mais importante dentro do ciclo de RBC e é neste contexto que se aplica a abordagem das abelhas dançando para a melhor fonte de néctar, com o objetivo de recuperar o melhor caso para o usuário.

Nessa abordagem, o modelo matemático de Camazine e Sneyd foi modificado e combinado com a abordagem de Raciocínio Baseado em Casos no desenvolvimento de um sistema de recomendação baseado em casos (chamado CASIS).

Neste sistema, cada abelha é um agente com as seguintes características:

- probabilidade de abandonar a fonte de néctar;
- probabilidade de recrutar mais abelhas pra dançar;
- probabilidade de continuar na mesma fonte.
- a lista de fontes que a abelha consegue ver.

Cada fonte de néctar é considerado um caso e o conjunto de fontes é a base de casos. A partir de uma nova consulta, as abelhas deixam a colméia a procura de fontes de néctar. Como mostra o modelo de Camazine e Sneyd a escolha de cada abelha é calculada através do uso de probabilidades. A mais importante é a probabilidade de abandonar a fonte, onde a abelha confere a qualidade da fonte de néctar e decide se continua ou não nesta fonte. Nesta adaptação do modelo, a abelha pode forragear em diversas fontes (e não apenas em duas como o modelo original).

Em um sistema tradicional de raciocínio baseado em casos, a nova consulta é comparada com todos os casos da base e o mais parecido é mostrado ao usuário. Mas no sistema CASIS, a similaridade é usada para auxiliar a abelha a tomar sua decisão de continuar ou não na fonte de néctar. Os casos com a menor distância (em relação a nova consulta) são considerados os mais parecidos, ou seja, as melhores fontes de néctar.

O experimento realizado com o sistema mostrou que utilizando a metáfora da dança das abelhas o sistema sempre retorna alguma informação para recomendar ao usuário, evitando a frustração do usuário em não obter recomendação do sistema.

Em anexo a este relatório, consta o artigo "‘Case-based recommender system inspired by social insects’", produzido como resultado das pesquisas realizadas durante a disciplina na área de Insetos Sociais aplicada a Sistemas de Recomendação. Este artigo foi submetido ao ICCBR 2005, 6th International Conference on Case-Based Reasoning. Nele podem ser consultados a descrição do sistema desenvolvido e os resultados obtidos no sistema.

3 SATISFAÇÃO DE RESTRIÇÕES

A aplicação de abordagens inspiradas pelo comportamento de colônias de insetos sociais na resolução de problemas discretos de otimização deu origem a uma área de pesquisa denominada *Ant Colony Optimization* - ACO. Os pesquisadores desta área buscam empregar uma meta-heurística, denominada *Ant System*, na resolução de diversos problemas deste tipo como o *Travel Salesman Problem* (TSP) e o *Quadratic Assignment* (QAP). Os algoritmos baseados no *Ant System*, conhecidos como *Ant Algorithms* vêm sendo aplicados com sucesso e em um número cada vez maior de problemas clássicos de otimização.

No artigo (TARRANT; BRIDGE, 2004) os autores discutem a aplicação de *Ant Algorithms* em problemas de satisfação de restrições. Primeiramente é apresentada a idéia básica do algoritmo. Em seguida, são discutidas e avaliadas as diversas abordagens para se obter a solução do problema usando este algoritmo básico. O objetivo dos autores foi determinar qual destas abordagens é a mais eficiente para no futuro comparar a abordagem baseada em insetos sociais com outras já apresentadas.

Os Problemas de Satisfação de Restrições (*Constraint Satisfaction Problems* - CSP) são caracterizados por conterem um conjunto de variáveis, cada uma possuindo um domínio específico (conjunto discreto de valores diferentes que a variável pode assumir) e um conjunto de restrições com relação a distribuição dos valores dos domínios entre as variáveis. O problema é considerado resolvido quando o menor número possível de restrições é desrespeitado. Este tipo de problema pode ser mapeado como um grafo onde os nodos representam as variáveis associadas com cada um de seus valores possíveis e os vértices representam as restrições existentes com relação aos pares variável/valor representados pelos nodos interligados pelo vértice em questão.

No algoritmo básico utilizado para os CSP uma quantidade inicial de feromônios é distribuída pelo grafo. Em seguida, formigas artificiais são aleatoriamente distribuídas nos nodos do grafo e se deslocam através dos nodos formando uma atribuição completa, isto é, percorrendo um nodo relacionado com cada variável. Ao final deste percurso são distribuídos feromônios de acordo com a qualidade da solução. Simultaneamente, os feromônios depositados evaporam segundo uma taxa pré definida. O algoritmo termina quando uma solução para o problema é encontrada ou quando um determinado número de percursos foi realizado.

A primeira variação possível para este algoritmo diz respeito a onde os feromônios serão depositados, se nos vértices ou nas arestas. Quando considerando o nodo, o feromônio depositado é proporcional ao custo do par variável/valor representado pelo nodo. Quando considerando os vértices, o feromônio depositado é proporcional ao custo do par variável/valor de ambos os nodos que são interligados pelo vértice em questão.

A maneira como as formigas virtuais selecionam o próximo vértice a ser visitado é determinado por uma regra de transição que depende de dois fatores: um heurístico e

outro baseado nos feromônios. O fator heurístico é um valor inversamente proporcional ao número de restrições violadas. O fator baseado nos feromônios pode ser considerando o depósito nas arestas ou nos vértices, como mencionado anteriormente.

Os fatores descritos acima podem ser combinados de duas formas: de forma composta, onde a formiga seleciona o vértice escolhendo a variável e seu valor simultaneamente através de uma equação que combina os fatores; em cascata, onde a variável é primeiramente selecionada através de uma heurística e, em seguida, o valor é selecionado do mesmo modo que na forma composta. A heurística para a seleção da variável pode ser uma das seguintes: *static-random*, onde as variáveis são ordenadas aleatoriamente no início do processo; *dynamic-random*, onde as variáveis são selecionadas aleatoriamente durante o percurso; *max-static-degree*, onde o próximo vértice a ser visitado é aquele que tem o maior número de restrições; *min-static-degree*, onde o próximo vértice a ser visitado é aquele que tem o menor número de restrições; *max-forward-degree*, onde o próximo vértice a ser visitado é aquele que está conectado a variável ainda sem valor que tem o maior número de restrições; *min-forward-degree*, onde o próximo vértice a ser visitado é aquele que está conectado a variável ainda sem valor que tem o menor número de restrições; *smallest-domain-first* onde o próximo vértice a ser visitado é aquele cuja variável tem o menor domínio.

Todas as combinações das abordagens acima aforam testadas no artigo. A combinação dos fatores heurístico e feromônios utilizando a abordagem composta se mostrou impraticável, consumindo muitos recursos. A combinação em cascata se mostrou mais eficiente, onde a heurística para a seleção da variável de melhor desempenho foi a *dynamic-random*.

O algoritmo básico apresentado e suas variações fogem bastante do objeto de sua inspiração: as colônias de formigas. Quando utilizando a combinação em cascata, que se mostrou mais eficiente, são considerados mais fatores aleatórios do que os feromônios. Além disso, a atualização do feromônio ocorrendo apenas no final do percurso e de uma só vez descaracteriza significativamente o processo natural utilizado pelas formigas.

A seguir o problema de satisfação de restrições será formalizado e uma das abordagens baseada em *Ant Algorithms* será discutida em detalhes. Na seção 3.3 será apresentado um trabalho realizado com problemas de satisfação de restrições que não tem relação com os insetos sociais. Este trabalho foi realizado, entre outras coisas, para se conhecer o estado da arte na solução de problemas de satisfação de restrições distribuídas. Este tipo de problema tem uma relação ainda maior com os insetos sociais que os problemas de satisfação de restrições centralizadas e serão considerados em um estudo futuro. Pretende-se estudar a possibilidade de desenvolver uma abordagem baseada em insetos sociais, provavelmente utilizando modelos de alocação de tarefas, para a solução destes problemas.

3.1 Formalização do CSP

Os Problemas de Satisfação de Restrições são problemas que têm diversas aplicações na vida real como: escalonamento, alocação de recursos, reconhecimento de padrões, etc. Um CSP pode ser definido pela tupla $\langle X, D, C \rangle$, onde: X é o conjunto finito de variáveis, D uma função de mapeamento e C é um conjunto de restrições.

O CSP pode ter somente restrições binárias, sendo chamado de *Random Binary CSP*. Uma classe de CSPs aleatoriamente gerados é caracterizada por 4 componentes: $\langle n, m, p_1, p_2 \rangle$, onde: n é número de variáveis, m é número de valores em cada domínio, $p_1 \in [0, 1]$ medida de conectividade (numero de restrições) e $p_2 \in [0, 1]$ número de pares de valores incompatíveis para cada restrição. Os experimentos do artigo (SOLNON,

2002) cuja abordagem será estuda a seguir foram todos obtidos com problemas de satisfação de restrições do tipo binário.

O problema também pode ser mapeado em um grafo não dirigido onde as arestas são as restrições e os nodos são os valores possíveis para as variáveis. Sendo um problema representado pelo grafo $G(V, E)$, $V = \langle X_i, v_i \rangle | X_i \in X \text{ e } v_i \in D(X_i)$ e $E = \{ \langle X_i, v \rangle, \langle X_i, w \rangle \in V^2 | X_i \neq X_j \}$.

3.2 Ant-Solver

Formigas artificiais podem ter capacidades extra não encontradas nos insetos reais. Na maioria dos casos o feromônio é liberado apenas depois de encontrar um caminho completo e a qualidade do caminho também pode ter influência na quantidade de feromônio. Inspirado nesses aspectos dos insetos sociais, o modelo proposto em (SOLNON, 2002) mostra um método de resolução de problemas de satisfação de restrições.

Algoritmo 1 Ant-Solver

Configura os parâmetros e inicializa os rastros de feromônio

repeat

for all k in $nbAnts$ **do**

 Constrói uma associação A_k

 Atualiza os rastros de feromônio usando $\{A_1, \dots, A_{nbAnts}\}$

end for

until $custo(A_i) = 0$ para algum $i \in \{1 \dots nbAnts\}$ **ou** o número máximo de ciclos foi alcançado

O *Ant-Solver*, mostrado no Algoritmo 1, funciona criando associações e construindo rastros de feromônio. A quantidade de feromônio em uma aresta é $\tau(\langle X_i, v \rangle, \langle X_j, w \rangle)$. Como o grafo não é dirigido: $\tau(\langle X_i, v \rangle, \langle X_j, w \rangle) = \tau(\langle X_j, w \rangle, \langle X_i, v \rangle)$. Para favorecer uma exploração maior e prevenir diferenças extremas entre dois caminhos, utiliza-se limites para o feromônio $0 < \tau_{min} < \tau_{max}$, do mesmo modo como foi proposto no sistema *MAX-MIN* (STÜTZLE; HOOS, 2000).

Algoritmo 2 Construção das associações

$A \leftarrow \emptyset$

while $|A| < |X|$ **do**

 Seleciona uma variável $X_j \in X$ que não tem associação em A

 Escolhe um valor $v \in D(X_j)$ com probabilidade $p_A(\langle X_j, v \rangle)$

$A \leftarrow A \cup \langle X_j, v \rangle$

end while

O *Ant-Solver* pode ter uma busca local, tornando-se um algoritmo híbrido que combina a construção probabilística da solução com a busca local. Cada vez que uma formiga constrói uma associação e antes de atualizar os rastros de feromônio, aplica-se a busca local para melhorar a associação construída. A cada passo escolhe-se aleatoriamente uma variável conflitante e escolhe-se um valor para essa variável que minimize o número de conflitos.

A introdução da etapa de pré-processamento, Algoritmo 3, é motivada pelo fato que as trilhas de feromônio tem alto custo de administração e começam a ter efeito na busca apenas depois de 100 ou mais ciclos. A idéia é criar um número significativo de associações

Algoritmo 3 Pré-processamento

Calcula $n_{Melhores}$ associações e guarda no *SampleSet*
repeat
 $CustoAntigo \leftarrow \sum_{A \in MelhorDe(n_{Melhores}, SampleSet)} custo(A)$
 calcula $n_{Melhores}$ associações a adiciona ao *SampleSet*
 $CustoNovo \leftarrow \sum_{A \in MelhorDe(n_{Melhores}, SampleSet)} custo(A)$
until $CustoNovo / CustoAntigo > 1 - \epsilon$ **ou** uma solução foi encontrada

utilizando uma busca local sem o uso de feromônios. Inicializa-se as trilhas de feromônio com as melhores associações encontradas. Para problemas simples, o problema é resolvido apenas com a fase de pré-processamento.

O Ant-Solver é um algoritmo genérico para resolver CSP utilizando ACO. Este algoritmo pode ser utilizado juntamente com busca local para atingir melhor desempenho. Além da busca local, pode ser incluída uma etapa de pré-processamento que aumenta a área de busca inicial do algoritmo e eleva o desempenho.

3.3 Satisfação de Restrições Distribuídas

Os Problemas da Satisfação de Restrições Distribuídas - *Distributed Constraint Satisfaction Problems* (DCSPs) - e da Otimização de Restrições Distribuídas - *Distributed Constraint Optimization Problems* (DCOPs) - estão sempre presentes em sistemas distribuídos que atuam em ambientes dinâmicos. Muitos problemas práticos considerados importantes em sistemas distribuídos como coordenação, escalonamento de tarefas e alocação de recursos podem ser formulados e resolvidos como DCSPs e DCOPs.

Mailler e Lesser (MAILLER; LESSER, 2004) apresentaram métodos baseados em mediação cooperativa para a resolução de DCOPs e DCSPs denominados *Optimal Asynchronous Partial Overlay* (OptAPO) e *Asynchronous Partial Overlay* (APO), respectivamente. Segundo os autores, estes protocolos apresentam soluções para os problemas em questão de uma forma rápida, distribuída e assíncrona sem o custo explosivo de comunicação que normalmente acompanha soluções deste tipo. A eficiência destes métodos foi experimentada através de sua aplicação em um problema de coloração de grafos, onde foi obtido um melhor desempenho em termos computacionais e de comunicação em relação a técnica mais eficiente atualmente aplicada, denominada Adopt (MAHESWARAN et al., 2004).

Um conhecido problema prático relacionado a otimização de restrições é o Problema do Agendamento Distribuído de Reuniões - *Distributed Meeting Scheduling Problem* (DMSP) (SEN; DURFEE, 1991) - onde se busca um agendamento para uma reunião visando maximizar o tempo útil dos participantes, mantendo a privacidade de suas informações. O DMSP envolve agendar uma reunião respeitando uma série de requisitos estabelecidos pelos participantes. Maheswaran et al. (MAHESWARAN et al., 2004) propuseram um mapeamento do DMSP para o DCOP, permitindo que algoritmos para a solução de DCOP sejam aplicados diretamente nos DMSPs. Estes autores utilizaram o Adopt para validar este mapeamento e verificaram que a aplicação do Adopt em DMSPs tem baixa eficiência em termos de custo de comunicação, número de iterações e do tempo necessário para se obter um agendamento. Algumas heurísticas foram propostas pelos autores que conseguiram melhorar o desempenho do Adopt neste cenário.

Os algoritmos para DCOP normalmente são validados em problemas de pequeno

porte, como o de coloração de grafos, onde o número de restrições e o tamanho do domínio do problema são pequenos. Maheswaran et al. mostraram que o melhor algoritmo conhecido (Adopt) não lidava com eficiência de DMSP. O objetivo deste trabalho foi aplicar o OptAPO na resolução de DMSP para analisar seu desempenho em cenários mais realistas, como foi feito com o Adopt, e comparar seus resultados com os obtidos pelo Adopt e o Adopt modificado (com as heurísticas mencionadas).

Uma vez que o OptAPO mostrou melhores resultados que o Adopt quando aplicado à problemas de coloração de grafos, era esperado o mesmo comportamento nos DMSPs. Contudo, o OptAPO não pode ser aplicado diretamente neste cenário dada a complexidade do problema, não convergindo para uma solução em um tempo polinomial. Com isso, o cenário inicialmente escolhido foi simplificado e a comparação pode ser realizada. O Adopt e o Adopt modificado se mostraram bem mais eficientes que o OptAPO com relação ao tempo de execução. Porém, nenhum dos dois foi capaz de superar o OptAPO em relação ao número de mensagens trocadas e o número de iterações necessários para a convergência. Com o objetivo de aproveitar estas qualidades do OptAPO, foi proposta uma alteração no algoritmo que substitui a garantia de optimalidade da solução por uma abordagem heurística mais eficiente quanto ao tempo de execução. Esta modificação mostrou bons resultados, fazendo com que o tempo de execução do OptAPO seja ainda menor que o do Adopt modificado.

Algumas questões ainda permanecem em aberto: é necessário analisar o desempenho da heurística proposta para o OptAPO em um conjunto grande de cenários para avaliar estatisticamente a qualidade das soluções obtidas. Além disso, é importante comparar esses resultados com outras abordagens que também não garantem a optimalidade (o Adopt possui algumas variações com essa característica). Finalmente, é preciso analisar mais detalhadamente as causas da ineficiência do OptAPO em DMSP e verificar se ela se repete em outros cenários igualmente complexos.

Este trabalho está melhor detalhado no artigo em anexo *Distributed Meeting Schedule through Cooperative Mediation: balancing optimality and performance* escrito por Paulo Ferreira e Ana Bazzan e que foi submetido para Nineteenth International Joint Conference on Artificial Intelligence 2005 - IJCAI.

4 CONCLUSÃO

O estudo realizado sobre insetos sociais revelou comportamentos interessantes e desejáveis para solução de problemas computacionais.

Uma das preocupações do grupo tratava-se da investigação de novas metáforas na área de Insetos Sociais para a aplicação em Sistemas de Recomendação Baseados em Casos, já que atualmente somente a metáfora de feromônio vem sendo usada e normalmente na área de otimização de problemas. Através dos resultados dos experimentos, a metáfora da dança das abelhas mostrou-se satisfatória no domínio de Sistemas de Recomendação pois o sistema sempre retorna uma recomendação para o usuário, evitando o seu desapontamento com sistemas de recomendação. Como trabalhos futuros, pretende-se explorar duas principais direções. Primeiramente, deseja-se aprofundar os estudos na área de Insetos Sociais, mais precisamente o comportamento das abelhas em ambientes altamente dinâmicos, ou seja, quando as pesquisas dos usuários variam rapidamente.

Outro interesse do grupo era em relação aos problemas de satisfação de restrições distribuídas. Foram estudadas as abordagens envolvendo os insetos sociais e sua aplicação em problemas de satisfação de restrições centralizadas. Além disso, foram estudados e analisados alguns dos melhores algoritmos para a satisfação/otimização de restrições distribuídas. Como trabalhos futuros nesta área, pretende-se estudar a possibilidade de desenvolver uma abordagem para os problemas de satisfação/otimização de restrições distribuídas baseada no modelo de alocação de tarefas dos insetos sociais que possa competir em desempenho com as abordagens estudadas atualmente.

REFERÊNCIAS

BESHERS, S. N.; FEWELL, J. H. Models of Division of Labor in Social Insects. **Annual Review of Entomology**, [S.l.], v.46, p.413–440, January 2001.

BURKE, R. A case-based approach to collaborative filtering. In: ADVANCES IN CASE-BASED REASONING: 5TH EUROPEAN WORKSHOP, EWCBR-2000, TRENTO, ITALY, SEPTEMBER 6–9, 2000: PROCEEDINGS, 2000. **Anais...** Springer, 2000.

CAMAZINE, S.; DENEUBOURG, J. D.; FRANKS, N. R.; SNEYD, J.; THERAULAZ, G.; BONABEAU, E. **Self-Organization in Biological Systems**. [S.l.]: Princeton, 2003.

CAMAZINE, S.; SNEYD, J. A Model of Collective Nectar Source Selection by Honey Bees: self-organization through simple rules. **Journal of Theoretical Biology**, [S.l.], v.149, n.4, p.547–571, April 1991.

MAHESWARAN, R. T.; TAMBE, M.; BOWRING, E.; PEARCE, J. P.; VARAKANTHAM, P. Taking DCOP to the Real World: efficient complete solutions for distributed multi-event scheduling. In: THIRD INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2004. **Anais...** [S.l.: s.n.], 2004. v.1, p.310–317.

MAILLER, R.; LESSER, V. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In: THIRD INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2004. **Anais...** IEEE Computer Society, 2004. p.438–445.

SEELEY, D.; CAMAZINE, S.; SNEYD, J. Collective Decision-Making in Honey Bees: how colonies choose nectar sources. **Behavioral Ecology and Sociobiology**, [S.l.], v.28, p.277–290, 1991.

SEN, S.; DURFEE, E. H. A Formal Study of Distributed Meeting Scheduling. In: CONFERENCE ON ORGANIZATIONAL COMPUTING SYSTEMS, 1991. **Anais...** [S.l.: s.n.], 1991. p.310–317.

SOLNON, C. Ants can solve constraint satisfaction problems. **IEEE Trans. Evolutionary Computation**, [S.l.], v.6, n.4, p.347–357, 2002.

STÜTZLE, T.; HOOS, H. H. MAX-MIN Ant system. **Future Gener. Comput. Syst.**, [S.l.], v.16, n.9, p.889–914, 2000.

TARRANT, F.; BRIDGE, D. When Ants Attack: comparing ant algorithms for constraint problems. In: PROCS. OF THE FIFTEENTH IRISH CONFERENCE ON ARTIFICIAL INTELLIGENCE AND COGNITIVE SCIENCE, 2004. **Anais...** [S.l.: s.n.], 2004. p.167–176.

APÊNDICE A

Case-based recommender system inspired by social insects

Fabiana Lorenzi^{1,2} Daniela Scherer dos Santos² and Ana L. C. Bazzan²

¹ Universidade Luterana do Brasil

Rua Miguel Tostes, 101

Canoas, RS, Brasil

lorenzi@ulbra.tche.br

² Instituto de Informática

UFRGS

Caixa Postal 15064

91.501-970 Porto Alegre, RS, Brasil

{dsradtke,bazzan}@inf.ufrgs.br

Abstract. Case-based recommender systems can learn about user preferences over time and automatically suggest products that fit these preferences. In this paper we present such a system, called CASIS. In CASIS we combined case-based reasoning approach with a metaphor from colonies of social insects, namely the honey bee dance. In nature, this is used to indicate the best nectar source among honey bees. Similarly we use it to retrieval the most similar case to the user's query. Our results show that this combination is effective when used in the retrieval step of the recommendation cycle: the most similar case is found by the "bees".

Keywords: Case-based recommender systems; social insects, swarm intelligence.

1 Introduction

Recommender systems are being used in e-commerce web sites to help the customers selecting products more suitable to their needs. The growth of Internet and eCommerce has brought the need for such a new technology [14], and a number of research projects have focused on these systems [13] applying different approaches.

One of these approaches, called knowledge-based recommender system, allows the system to learn about user preferences over time and automatically suggest products that fit the learned user model. This paper focuses on case-based recommender systems which is classified as a knowledge-based approach, showing the CASIS system, a combination of two approaches: case-based reasoning and swarm intelligence through the honey bees dance metaphor. The former allows to retrieve solution cases and use them to solve new problems. The latter approach is inspired by social insects: honey bees dance to indicate the best nectar source. This combination would still be classified as a case-based

recommender system according to the unifying framework presented in [10]. The motivation of combining these approaches is to improve the recommendation to the user.

The next section provides a brief introduction to case-based recommender systems and to swarm intelligence based approaches. Section 3 describes the proposed approach and section 4 shows the experiment results. Finally section 5 shows the conclusions and the future work.

2 Related Work

2.1 Case-Based Recommender Systems

Case-Based Reasoning (CBR) is a problem solving methodology that deals with a new problem by first retrieving a past, already solved similar case, and then reusing that case for solving the new problem [1]. In a CBR recommender system (CBR-RS) a set of suggested products is retrieved from the case base by searching for cases similar to a case described by the user [4].

CBR can support the recommendation process in a number of ways. In the simplest approach, the CBR retrieval is called taking as input a partial case defined by a set of user preferences (attribute-value pairs), and a set of products matching these preferences are returned to the user. In this process we can identify some basic tasks such as the input (where the user provides his/her requirements), the products retrieval (where the system searches for products according to user requirements), and the output, where some recommendation is given to the user.

In the simplest application of CBR to recommendation, the user is supposed to look for some product to purchase. He/she inputs some requirements about the product and the system searches the case base for similar products (by means of a similarity metric) that match the user requirements. A set of cases is retrieved from the case base and these cases can be recommender to the user. If the user is not satisfied with the recommendation he/she can modify the requirements, i.e. build another query, and a new cycle of the recommendation process is started.

The case retrieval is typically the main step of the CBR cycle and the majority of CBR recommender systems can be described as sophisticated retrieval engines. For example, in the Order-Based Retrieval [3] the system uses special operators to retrieve a lattice of cases, or in the Compromise-Driven Retrieval [12] the system retrieves similar cases from the case base but also groups the cases, putting together those offering to the user the same compromise, and presents to the user just a representative case for each group.

2.2 Swarm intelligence

The use of the social insect metaphor to solve computer problems such as combinatorial optimization, communications networks or robotics is increasing [2].

Social insects living in colonies, e.g. ants, termites, bees, and wasps [18] distinguish themselves by their organization skill without any centralized control [5,

9]. Organization emerges from interactions among individuals, between the individuals and the environment, and from behaviors of the individuals themselves [2].

Several approaches exist; the complete list is outside the scope of this paper. However, the following works are related to our approach: [8, 16]. The former has implemented and compared six Ants Colony Optimization (ACO) algorithms [7] to solve constraint satisfaction problems.

This section describes swarm intelligence and specifically self-organization among honey bees. Among these, the colony selects the best nectar source available through simple behavioral rules. The process of dispatching bees into the surrounding countryside to gather the colony's food is called foraging. Bees travel up to 10km from the hive to collect nectar. They return with nectar and information about the nectar source [5].

The bee has three behavior options in the foraging process [5]:

1. to share the nectar source information by dancing, a behavior in which a bee communicates to other bees the direction, distance, and desirability of the food source, trying to recruit new bees to that food source.
2. to continue foraging without recruiting other bees.
3. to abandon the food source and go to the area inside the hive called the dance floor to observe dancing bees and select its next food source.

In [15] the authors presented an experiment showing the recruitment of nectar forager bees to feeders containing sugar solutions. Two food sources are presented to the colony at 8:00 a.m. at the same distance from the hive: source A is characterized by a sugar concentration of 1.00 mol and source B by a concentration of 2.5 mol. At the noon, the sources are exchanged: source A is now characterized by a sugar concentration 2.5 mol and source B by a concentration of 0.75 mol. In both situations the better source is more visited. This experiment showed that the foraging decision of each bee is based on very limited information of its own particular visited source. This simple behavior allows the colony to select the best quality source of nectar.

Based on these observations, colonies select the best quality source through the rate of dancing and abandonment based upon nectar source quality. Camazine and Sneyd [6] developed a mathematical model which demonstrates how the properties of the system emerge automatically from the dynamic interactions among the constituent components.

The recruitment of other bees to forage a good nectar source is considered a positive feedback in the honey bees organization [17]. The positive feedback is also a step of the case-based recommender system's cycle and its importance is well recognized in many researches (see [11] for more details).

In the next section we show how the metaphor from honey bees behavior was applied to the development of case-based recommender systems.

3 The CASIS system

3.1 The mathematical model

The CASIS system is based on a mathematical model described by Camazine and Sneyd [6] which demonstrates how the properties of the system emerge automatically from the dynamic interactions among the bees. To validate this mathematical model, the authors considered two nectar sources (A and B). Figure 1 shows the flow diagram with the scenerio. It is divided in seven compartments where the bees can follow one of the behaviors:

- H_A and H_B : unload nectar from food source A and B, respectively;
- D_A and D_B : dance for food source A and B, respectively;
- A and B: forage at food source A and B, respectively;
- F: follow a dancer (after having watched the dancer).

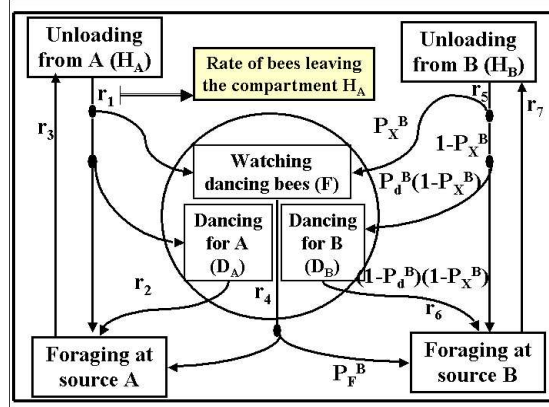


Fig. 1. Camazine and Sneyd's mathematical model [6]

Two factors affect the proportion of the total forager force in each of the seven compartments: the rate at which a bee moves from one compartment to another r_{1-7} , and a probability that a bee takes one or other fork at five branch points. These forks are presented in figure 1 as black points. We give the formulas only for source B, for the sake of example.

After the bee unloaded the nectar in the hive, it finds the first decision fork: abandon the source B (P_X^B) or continue in the source B ($1 - P_X^B$). If it chooses to continue in the source, then it can decide also if it recruits new bees for this source ($P_d^B(1 - P_X^B)$) or not. If the bee has decided to abandon the source it can watch the dancers and decide which one it will follow for nectar source (P_F^B). This probability is estimated by $B(P_F^B) = \frac{D_B d_B}{D_B d_B + D_A d_A}$. where:

- d_A and d_B are the proportions of time that the foragers actually dance; it must be said that only a portion of a bee's time in the dance area is actually spent on dancing, and that the dancing time is proportional to the quality of the nectar source (thus the best the source, the higher the chance that other bees will observe the dance and go to that source);
- D_A and D_B are the number of bees in the source A and B respectively.

Summarizing, the probabilities are:

- P_X^A and P_X^B : probabilities of abandoning A and B respectively, per foraging trip;
- P_d^A and P_d^B : probabilities of dancing for A and B respectively;
- P_F^A and P_F^B : probabilities of following a dancer for A and B respectively.

3.2 The approach

In our approach, Camazine and Sneyd's mathematical model was adapted and combined with case-base reasoning to a case based swarming intelligence recommender system (called CASIS).

In the simulation, each bee is an agent with the following features:

- probability of abandoning the nectar source;
- probability of recruiting more bees by dancing;
- probability of keep following the source;
- the list of sources that the bee is able to see.

Each nectar source is considered a case and the set of sources is the case base. Given a new user query, the metaphor calls for honey bees leaving the nest looking for nectar sources. As shown in Camazine and Sneyd's model [6], each bee's choice is calculated by using probabilities. The most important is the abandoning's probability where the bee checks the quality of the nectar source and decides whether or not to continue in this source. In our system the original model was modified and the bee can forage for several different sources (not only two as in the original model).

Figure 2 shows the adapted model with the hive having two compartments (dancing or observing), the case base (representing the available sources), and the diamonds representing the bee's decisions possibilities. To decide what to do, some probabilities are calculated:

- P_x^i : probability of abandoning case i , per foraging trip;
- P_d^i : probability of dancing for case i ;
- P_F^i : probability of following a dancer for case i .

The probability of abandoning a case was adapted in our model. In a traditional case-based recommender system, the new query is compared with all cases in the case base and the most similar is shown to the user. But in CASIS, the similarity is used to help the bee to make its decision to continue in the nectar source or not. The P_x^i is the distance between the new query and the cases. The

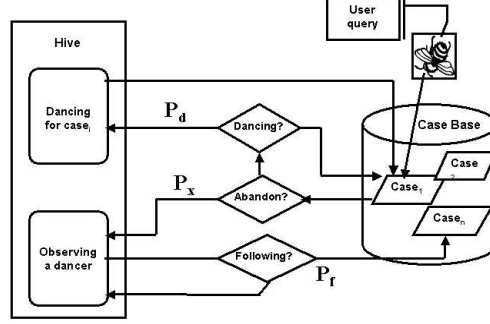


Fig. 2. CASIS's approach – adapted from Camazine and Sneyd

cases with the smallest distance to the new query are the most similar, i.e. they represent the best nectar sources.

The probability of following a dancer was also adapted because we are not taking into account the time the bee spend dancing. Thus, $P_F^i = \frac{D_i}{\sum_{j=1}^n D_j}$ where D_i is the number of dancers bees to the source i and n is the total number of bees.

3.3 Validation

One scenario from tourism was used to validate the approach. The recommender cycle starts with a simple query where the user specifies some preferences. The next step is the retrieval. The bees start to look for the best cases and after a number of iterations the best one is recommended to the user. The review step was implemented in a different way here. Normally, a recommender system allows the user to refine the query and to start a new recommendation cycle. However in this experiment we use the honey bees feedback to search cases to recommend, meaning that one recommendation cycle is composed by many iterations. It is important to notice that the probabilities regarding each bee are recalculated in each iteration, using the similarity between the case and the query.

The bees' behavior always allow them to recommend something to the user thus avoiding empty recommendations. This is due to the fact that the system always recommend the best nectar source, i.e. the case which is visited by the highest number of bees. The revise and retain steps were not implemented in this experiment.

4 Results

The implementation was done using the SeSAm tool³ for agent-based simulation.

In our model the parameters of the simulation are the number of cases, the number of bees, and the number of iterations. Although we have ran the system with different sets of parameters, the case base was the same in each experiment. It has 100 cases and each case represents a tourism package from Brazilian tourism operators. Attributes of the cases (packages) are: destination, length, and price.

In the first experiment, we used 2500 bees (which is considered a high number) and the system was allowed to run until the best case emerges. Figure 3 shows the results of this first experiment. We plot only the most visited cases during the simulation in order to avoid a dense graphic. Among these, only three have high similarity with the user query. The figure shows that, in the beginning, bees consider various cases because they have little information about them (as it is the case with real nectar sources which are visited randomly). By time $t=20$ the best case emerges as *case 04* and this is in fact the best package the user can have (in the case base, *case 04* is the one with the highest similarity to the user's query).

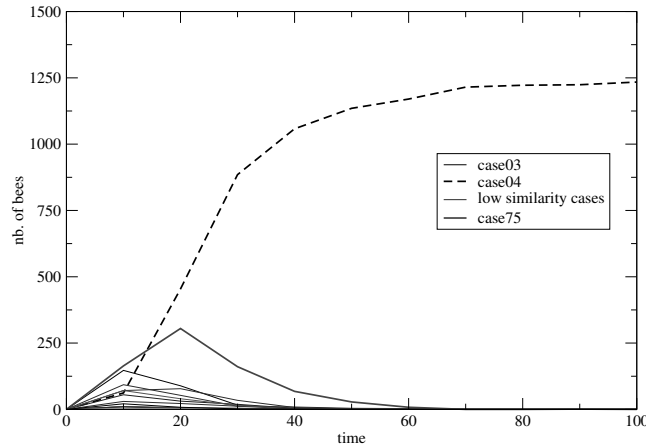


Fig. 3. Number of bees visiting each case along time: experiment with 2500 bees

In the next experiment shown in figure 4 we used the same user query but we reduced the number of bees to 500, allowing the simulation to run up to

³ available for download at <http://www.simsesam.de>

500 iterations. Here, an interesting situation arises: bees started to visit *case 69* because it had the highest similarity *in the beginning*. However, as soon as *case 04* was founded as a better case (around $t=150$), bees start abandoning visiting *case 69* and prefer to visit *case 04*. This shows that the approach is effective.

Moreover, the approach is also robust to dynamic changes in the queries: during the iterations, it is possible to notice that the bees' behavior is automatically clustered around cases according to the user's preferences. For example, in the tourism scenario, users' preferences usually change with seasons of the year: during summer beaches are favorite spots, whereas in winter people normally look for skydiving or tropical resorts. Once the user's query changes, so does the clustering of bees.

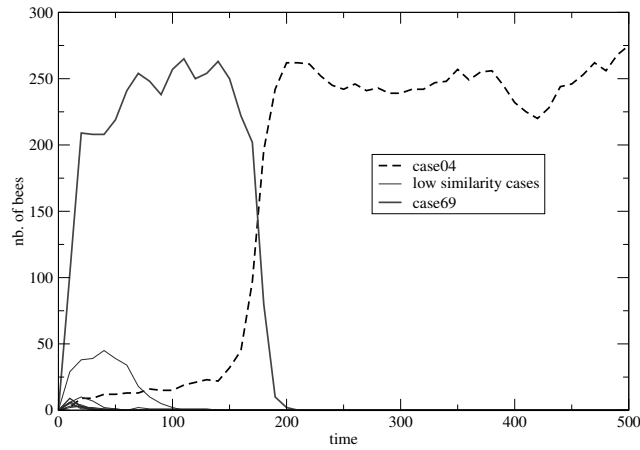


Fig. 4. Number of bees visiting each case along time: experiment with 500 bees

5 Conclusions and Future Work

This paper presented the CASIS system, a case-based recommender systems that uses swarm intelligence in the retrieval step. This use is justified by the success social insect have in dynamic changing environments.

Normally only the pheromone metaphor is used, which is suited for optimization problems. The motivation of our work was to explore a different and more suitable metaphor for CBR: the honey bees dance. The positive feedback of the social insects can be considered as a common point between the bee's behavior and the case-based recommender systems.

Our experiments' results have shown that using this metaphor the system always return something to recommend to the user, avoiding the user's disappointment with the recommender system.

In the future we intend to explore two main directions. First, we intent to better study the behavior of the bees (e.g. varying their number) in highly dynamical environments, meaning that user's queries change rapidly. We are particularly interested in how robust the system is and how many bees are necessary to guarantee this robustness.

As another future work we want to improve the case representation, saving the previous queries as cases. Depending on the similarity of the new query the bees can start not completely randomly but using information gathered regarding the previous query.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
2. E. Bonabeau, G. Thraulaz, and M. Dorigo. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford Univ Press, 1999.
3. D. Bridge and A. Ferguson. Diverse product recommendations using an expressive language for case retrieval. In S. Craw and A. Preece, editors, *Advances in Case-Based Reasoning, Proceedings of the 6th European Conference on Case Based Reasoning, ECCBR 2002*, pages 43–57, Aberdeen, Scotland, 4 - 7 September 2002. Springer Verlag.
4. R. Burke. Knowledge-based recommender systems. In J. E. Daily, A. Kent, and H. Lancour, editors, *Encyclopedia of Library and Information Science*, volume 69. Marcel Dekker, 2000.
5. S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton, 2003.
6. S. Camazine and J. Sneyd. A model of collective nectar source selection by honey bees: Self-organization through simple rules. *Journal of Theoretical Biology*, 149(4):547–571, April 1991.
7. M. Dorigo, G. Di Caro, and L. Gambardella. Ant colony optimization: A new meta-heuristic. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalazala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.
8. T. Finbarr and D. Bridge. When ants attack: Comparing ant algorithms for constraint problems. In *Fifteenth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 167–176, 2004.
9. D. Gordon. The organization of work in social insect colonies. *Nature*, 380:121–124, 1996.
10. F. Lorenzi and F. Ricci. Case-based reasoning and recommender systems. Technical report, IRST, 2004.
11. L. McGinty and B. Smyth. Comparison-based recommendation. In S. Craw and A. Preece, editors, *Advances in Case-Based Reasoning, Proceedings of the 6th European Conference on Case Based Reasoning, ECCBR 2002*, pages 575–589, Aberdeen, Scotland, September 2002. Springer Verlag.

12. D. McSherry. Similarity and compromise. In A. Aamodt, D. Bridge, and K. Ashley, editors, *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, pages 291–305, Trondheim, Norway, June 23–26 2003.
13. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings ACM Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.
14. J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.
15. D. Seeley, S. Camazine, and J. Sneyd. Collective decision-making in honey bees: How colonies choose nectar sources. *Behavioral Ecology and Sociobiology*, 28:277–290, 1991.
16. C. Solon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6:347–357, 2002.
17. P. Tarasewich and P. McMullen. Swarm intelligence: Power in numbers. *Communications of the ACM*, 45(8):62–67, 2002.
18. E. O. Wilson. *Sociobiology: The New Synthesis*. Harvard Univ Press, 2000.

APÊNDICE B

Distributed Meeting Schedule through Cooperative Mediation: balancing optimality and performance

Keywords: Multiagent Systems and (Distributed) Constraint Satisfaction/Optimization

Abstract

In multiagent systems, Distributed Constraint Optimization Problem (DCOP) has been used as a formalism to model a wide range of agents' coordination issues. An approach based on cooperative mediation was recently proposed as a new way to find the optimal solution to DCOP. An important question about any solution to a DCOP is whether it is fast enough to be applied in real-world applications, such as Distributed Meeting Scheduling problems. Here, we map the DMS as a DCOP, use cooperative mediation, and compare the performance of this approach with the results achieved by another algorithms for DCOP. Given the time complexity of the complete solutions, we propose a modified approach for the cooperative mediation, in which the idea is to trade the completeness of the search mechanism for the performance of a heuristic search, which yields a good solution in a feasible time.

1 Introduction

The multiagent paradigm has coordination as a central issue. Coordination is a process in which agents engage to ensure that a community of individual agents acts in a coherent manner; when distributed agents work towards a common goal they should act as an unit coordinating interdependent actions, minimizing redundant efforts, sharing resources, etc. Research on coordination aspects has motivated many approaches and methods. Despite these many possibilities, no one is universally accepted as a complete solution. Thus, the literature presents some classical problems in agents' coordination such as distributed resource allocation problems, distributed scheduling problems, distributed planning problems, and so forth.

Distributed Constraint Optimization Problem (DCOP) is a formalism to model a wide range of the classical problems mentioned above. It is a distributed version of constraint optimization problems, which derived from constraint satisfaction problems. Differently from COP, in DCOP collaborative agents must find solutions over a distributed set of constraints. A DCOP consists of n variables $V = \{x_1, x_2, \dots, x_n\}$ that

can assume values from a discrete domain D_1, D_2, \dots, D_n respectively. Each variable is assigned to one agent that has the control over its value. The goal of the agents is to choose values for the variables to optimize a global objective function. This function is described as the sum over a set of valued constraints related to pairs of variables. Thus, for a pair of variables x_i, x_j , there is a cost function defined as $f_{ij} : D_i \times D_j \rightarrow N$. DCOP generalizes the Distributed Constraints Satisfaction Problem (DisCSP) [Yokoo *et al.*, 1998], which has a limited power of representation since every constraint is required to be boolean. This requirement is inadequate to represent many real-world problems where several degrees of quality or cost are necessary. Besides, real problems are often over-constrained, meaning that is impossible to satisfy all constraints.

In [Modi *et al.*, 2003] an asynchronous complete method for distributed constraint optimization (called Adopt) is proposed to find the optimal solution for problems formalized as DCOP. Adopt provides quality/optimality guarantees on system performance and asynchrony on communication among agents. Besides, Adopt is known as the most efficient algorithm for DCOP [Maheswaran *et al.*, 2004]. Modi *et al.* discuss the main differences between Adopt and previous approaches, showing its achievements and limitations.

Another approach, this time based on cooperative mediation, was recently proposed by Mailler and Lesser [Mailler and Lesser, 2004] as a new way to find the optimal solution to DCOP. Cooperative mediation is a partial centralization technique and it is the basis of the *optimal asynchronous partial overlay* (OptAPO) algorithm. OptAPO is also a complete, distributed algorithm to solve DCOP. Furthermore, the authors show that OptAPO performs better than Adopt in an abstract problem of graph coloring that is an instance of the MaxSAT problem.

An important question about all solutions to DCOP is whether the proposed algorithms are fast enough to be applied in real-world applications. An important question here is whether the number and size of exchanged messages makes the approach feasible. In distributed approaches, the communication among agents usually poses demands that can cause network overload. Is the total time consumed acceptable in these situations? Real-world problems usually mean that the planning (for coordination) and action should be treated as quickly as possible. Most of the proposed approaches yields

good results in simple scenarios, but there is a lack of analysis in complex real-world ones.

One of these scenarios is the Distributed Meeting Schedule problem (DMS) [Sen and Durfee, 1991]. In a DMS, a group of persons wish to attend several meetings. The attendees try to optimize their calendars according to personal preferences maintaining the privacy of their information. Each meeting is subject to many constraints. The negotiation proxies to produce a schedule can incur in high communication costs and unacceptable time.

In [Maheswaran *et al.*, 2004], a DMS is mapped to a DCOP using a systematic reusable framework called Distribute Multi-Event Scheduling (DiMES). Agents' goal is to generate a coordinate schedule for the execution of joint activities or resource usage in a multiple-events scenario. Besides presenting DiMES, the authors have also tested the efficiency of Adopt in some real-world problems mapped as DCOP using DiMES, and presented two heuristics to improve Adopt's performance. They show that the convergence time of Adopt in the tested scenarios was one hundred times higher than expected, illustrating the existence of a significant difference in Adopt's performance between simple abstract and complex real-world problems. The presented heuristics reduce the convergence time to the expected values and could potentially make Adopt able to deal with complex problems.

In the present paper we discuss the difficulties of applying the cooperative mediation (OptAPO) algorithm in a real-world problem. To illustrate this, we use the Distributed Meeting Scheduling problem mapped as a DCOP using the DiMES framework. The goal here is twofold. First, to check how OptAPO handles a real-world DMS problem, given that the evaluation of OptAPO was initially based on an abstract scenario. We compare its performance with the results achieved by the Adopt, with and without heuristics. Second, we propose to change OptAPO's centralized search mechanism. The idea is to trade the completeness of search (which is achieved via branch-and-bound) for the performance of a heuristic search (using a genetic algorithm). The motivation behind this trade-off is that in real-world applications such as meeting scheduling, to have a good solution in a short time is better than achieving the optimal solution in a long time.

In order to show these points, in Section 2 we summarize the cooperative mediation approach and explain how the OptAPO works. In Section 3 we describe the DMS problem and its mapping to a DCOP using DiMES. In section 4 we discuss the OptAPO's performance and show the results, comparing them to those achieved by Adopt and a centralized approach based on branch-and-bound (B&B). Given the performances, also in Section 4 we propose to replace the B&B by a genetic algorithm (GA). Finally, in Section 5 we conclude giving further directions for this work.

2 Cooperative Mediation and the OptAPO

The optimal asynchronous partial overlay (OptAPO) [Mailler and Lesser, 2004] is a cooperative mediation based DCOP protocol. As said, OptAPO (as well as Adopt) is a complete method, meaning that its final solution is the optimal one. The algorithm allows the agents to extend the context they use for

local decision-making to a *relationship graph*. Within this graph, one of the agents has to act as a mediator, computing a solution for this extended context and recommending values for the variables associated with the agents involved in the mediation session. This is possible because agents construct a *good_list* – which holds the names of agents that have direct or indirect relationship to the list owner – and an *agent_view* – to hold the names, values, domains, and functional relationships of related agents.

During the problem-solving process, each agent tries to improve the value of its subproblem (the one it can solve within its relationship graph). The priority to take the mediation role is given to the agent with more information about the problem. A connected graph models the DCOP where each node is an agent (plus its values), and the links are the problem constraints. Each constraint or functional relationship has an associated cost. The algorithm has three stages: initialization, checking the agent view, and mediation. Details of these stages can be found in [Mailler and Lesser, 2004]; we give here a brief description. During the initialization, the agent sets its variables: current value (d_i), variable's name (x_i), priority (p_i), domain (D_i), functional relationships (C_i), *good_list* and *agent_view*.

The agent's goal is to improve the solution for its subproblem (represented by F_i). Thus, during the second stage, the *agent_view* is used to calculate the current cost F_i within the relationship subgraph given by i 's *good_list*. If $F_i > F_i^*$ (F_i^* being the optimal value of the subsystem), then agent i conducts either a passive or an active mediation session, after which the value of F_i^* is recomputed. Agent i sets a passive mediation if its priority to mediate is lower than the priority of another agent in the subsystem; otherwise it sets a temporary mediation flag (m_i') to active. If an active mediation flag is on, the agent can actually mediate only if there is no other agent with a higher priority to mediate and with an active mediation flag. The agent tests if a change in its local value would cause a local cost to reach the optimal cost. If it does, then the agent changes its value and does not start the mediation process. If the agent is has a passive mediation flag, it starts a passive mediation process.

In the mediation stage, agents receiving a mediation request either evaluate it or send a wait message. Evaluation means looking at each of the domain elements, labelling them with the names of the agents which share *functional relationships* with cost $f_i > f_i^*$, and returning these in a message. The mediator conducts a branch-and-bound (B&B) search to minimize the cost of the subproblem in its *good_list*, as well as the costs for agents outside the mediation session.

In [Mailler and Lesser, 2004] the OptAPO algorithm was applied to the MaxSAT 3-coloring problem with assignments for different number of variables (agents): 8, 12, 16, etc. Two series of tests were run with under- and over-constrained instances of the problem. These experiments compute the total number of messages, cycles, and time consumed to achieve the solution (measured in seconds using a standard PC configuration). Two conclusions of this evaluation should be pointed out here: OptAPO outperforms Adopt in terms of cycles (a round of message changing among agents), messages and runtime; and the OptAPO runtime is not a byproduct of

the centralized search (B&B). In Section 4 we show that in a more complex scenario (DMS problem) these conclusions could not be observed.

3 Distributed Meeting Scheduling as a DCOP

In Distributed Meeting Scheduling (DMS) a group of persons wishes to attend several meetings that should be scheduled in a distributed fashion. The schedule is built interactively by a cooperative network of decision-makers. When doing this within a multiagent system, each agent normally has knowledge only about the meetings it participates and its preferences concerning the schedules. The agents must negotiate to build a consistent global schedule that fits the attendees agenda and respects individual preferences. Additionally, in real-world scenarios of DMS, the attendees try to optimize their agendas according to personal preferences maintaining the privacy of their information. As said, all this negotiation process may produce high communication cost and take an unacceptable time.

In DiMES, a DMS problem is formalized as follows:

- the group of attendees (agents) are represented as a resource set $\mathcal{R} = \{R_1, \dots, R_n\}$ of cardinality N where R_n refers to the n -th resource (attendee);
- the agenda is represented by a fractionated time interval. Let $T \in \mathbb{N}$ be a natural number and Δ be a length such that $T \cdot \Delta = T_{latest} - T_{earliest}$. The possible time intervals to schedule the meetings (time domain) are represented by the set $\mathcal{T} = \{1, \dots, T\}$ of cardinality T where $t \in \mathcal{T}$ refers to the time interval $[T_{earliest} + (t-1)\Delta, T_{earliest} + t\Delta]$;
- the meetings are represented as an event set $\mathcal{E} = \{E^1, \dots, E^K\}$ of cardinality K where E^k refers to the k -th event. The k -th event E^k is characterized as the triple $(A^k; L^k; V^k)$ where $A^k \subset \mathcal{R}$ is the subset of resources that are required by the event, $L^k \in \mathcal{T}$ is the number of contiguous time slots necessary to schedule the event, and V^k is the vector of preferences of each resource to each event as described next;
- the preferences of an attendee is represented by a vector V^k whose length is the cardinality of A^k . If $R_n \in A^k$, then V_n^k is an element of V^k denoting the value per time slot of the n -th resource to schedule event (meeting) k . The n -th resource also has a value $V_n^0(t) : \mathcal{T} \rightarrow \mathbb{R}^+$ to keep the time slot t free of meetings.

In summary, the DMS problem is about how to distribute the meetings through the agendas' time slots in order to maximize the attendees preferences. Let us define a schedule S as a mapping from the event set to the time domain where $S(E^k) \subset \mathcal{T}$ denotes the time slots committed for event k . All resources in A^k must agree to allocate time slots $S(E^k)$ to event E^k . So, formally the problem, which is NP-hard, is: $\max_s (\sum_{k=1}^K \sum_{n \in A^k} \sum_{t \in S(E^k)} (V_n^k - V_n^0(t)))$ such that $S(E^{k_1}) \cup S(E^{k_2}) = \emptyset \forall k_1, k_2 \in \{1, \dots, K\}, k_1 \neq k_2, A^{k_1} \cap A^{k_2} = \emptyset$.

In [Maheswaran *et al.*, 2004], it is shown how to convert a DMS problem to a DCOP in which the goal is to optimize a global objective function. This function is described

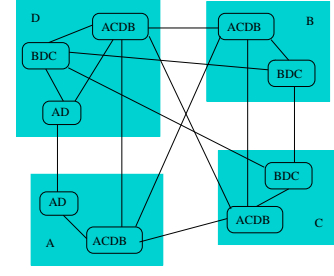


Figure 1: DCOP graph using PEAV approach

as the sum over a set of valued constraints related to pairs of variables. Usually, a DCOP is represented by a graph, where nodes represent the set of variables and the edges represent the utility function determined by the values of the adjacent nodes. For each edge $e(i, j) \in E$ there is a function $f_{ij}(x_i, x_j) : D_i \times D_j \rightarrow \mathbb{R}$. To solve the problem, one assignment $a^* \in A = D_1 \times \dots \times D_N$ must be chosen, such that $a^* = \text{argmax}_{a \in A} \sum_{(i,j) \in E} f_{ij}(x_i = a_i, x_j = a_j)$.

According to Maheswaran *et al.* there are three possible approaches for the DMS-DCOP mapping:

- time slots as variables (TSAV): in the TSAV each variable $x_n(t)$ represents a time slot (t) of a resource (n), which means $T \times N$ variables. The domain of this variables are the set of events, and all variables having the same resource belongs to the same agent. This approach leads to a very dense graph;
- events as variables (EAV): in the EAV each variable represents the start time x^k of an event E^k . The domain of these variables is the set of time slots which is early enough so that the event can be accommodated. Each variable is assigned to one of the agents that attends the event. This approach results in a simple graph. Furthermore, the agents make decisions about multiple resources. To make this decision, the information about preferences must be shared among agents. In real-world scenarios, where an agent represents a resource (attendee), this approach is not realistic due to the issue of lack of privacy;
- private events as variables (PEAV): the PEAV is an alternative to tackle the privacy problem of EAV. The main idea is the same of EAV (variables representing events). However, the agents make decisions only about the events they participate. Let us consider a set of variables $X^k = \{x_n^k : n \in A^k\}$ where $X^k \in \{0, 1, \dots, T - L^k + 1\}$ denotes starting time for event E^k where the resource R_n takes part. The DCOP is constructed with the set of variables $X = \cup_{k=1}^K X^k$. Each variable $X_n \in X$ that represents the n -th resource belongs to the same agent. These variables are fully connected (intra-agent links). The constraint utility function was designed to compute the values of internal links so as to maintain the privacy. There are inter-agents links connecting all participants of each event.

We are interested in representing DMS problems as close as possible to reality. For this reason we prefer the PEAV ap-

proach. Next we show the constraint utility function of PEAV and a conversion example. For details of each approach, including the variable sets, the utility functions, and the proofs of congruency, please refer to [Maheswaran *et al.*, 2004].

The resource constraint utility function used in the PEAV approach between the variables $x_{n_1}^{k_1}$ and $x_{n_2}^{k_2}$, when $x_{n_1}^{k_1} = t_1$ and $x_{n_2}^{k_2} = t_2$, is given by $f(n_1, k_1, t_1; n_2, k_2, 2_1)$, where:

$$f(n_1, k_1, t_1; n_2, k_2, 2_1) = -MI_{\{n_1 \neq n_2\}}I_{\{k_1 = k_2\}}I_{\{t_1 \neq t_2\}} + I_{\{n_1 = n_2\}}I_{\{k_1 \neq k_2\}}f_{intra}(n_1, k_1, t_1; k_2, t_2) \quad (1)$$

where

$$f_{intra}(n_1, k_1, t_1; k_2, t_2) = \begin{cases} -M & t_1 \neq 0, t_2 \neq 0, t_1 \leq t_2 \leq t_1 + L^k - 1, \\ -M & t_1 \neq 0, t_2 \neq 0, t_2 \leq t_1 \leq t_2 + L^k - 1, \\ g(n, k_1, t_1; k_2, t_2) & \text{otherwise} \end{cases}$$

and

$$g(n, k_1, t_1; k_2, t_2) = \frac{1}{|X_n| - 1} (Z_n^{k_1}(t_1) + Z_n^{k_2}(t_2))$$

where

$$Z_n^{k_i}(t_i) = \sum_{l=1}^{L^{k_i}} (V_n^{k_i} - V_n^0(t_i + l + 1))I_{\{t_i \neq 0\}}$$

with $M > NTV_{max}$ where N is the number of participants, T is the number of time slots and $v_{max} = \max_{k,n} V_n^k$.

As an illustrative example, assume that four attendees $\{A, B, C, D\}$ should schedule three meetings $\{E^1, E^2, E^3\}$ in a 3-time-slot agenda. Each meeting takes one time-slot and has the following configuration: E^1 involves A and D, E^2 A,C,D, and B, and E^3 involves B,D, and C. Figure 1 shows the DCOP graph for this example.

4 DMS through Cooperative Mediation

To apply OptAPO in the DMS problem we implemented a simple simulator using Java and conducted experiments with randomly generated instances of the DMS problem. We have used the same scenarios used in [Maheswaran *et al.*, 2004] to test the Adopt algorithm in real-world problems, mapping them as DCOPs using DiMES, following the PEAV approach. We ran the experiments in a standard PC under Linux.

Let us start discussing the performances of both OptAPO and Adopt in a simple scenario (MaxSAT 3-coloring problem, i.e. the domain size is 3). Although both were evaluated also with a large number of agents (more than 20), the density of the graph and the size of the domains were restricted. They worked with a number of constraints equal to $2 \times K$ and $3 \times K$, where K is the number of nodes. Despite this, the evaluation of Adopt has shown that more than 10,000 cycles were necessary to reach the optimal solution. They have also shown that approximately 50 messages were exchanged among the agents in each cycle. Although the total time (in seconds) has not been analyzed, it is reasonable to expect this to be high. We can see in these results that to find an optimal solution is expensive even in simple scenarios. The total

number of exchanged messages exceeds 500,000, which may represent a significant overload in a communication network. It is important to point out that the performance of Adopt was compared to a centralized search based on B&B, i.e. the same centralized search used in OptAPO. Adopt outperforms B&B in this analysis.

The evaluation of OptAPO in the same scenario has shown an immense reduction in the number of messages and cycles. The total number of exchanged messages does not exceed 15,000 and the number of cycles does not exceed 120. OptAPO outperforms Adopt in this analysis, proving that, in simple scenarios, the OptAPO runtime is not a byproduct of the centralized search (B&B).

Let us consider a more complex, real-world scenario: a DMS problem. In this scenario 9 attendees, with a 8-time-slot agenda, must schedule 8 meetings. This scenario was randomly chosen the data set in [Maheswaran *et al.*, 2004]. Converting this DMS problem to DCOP (using PEAV) generates 23 variables, with a 8-element domain, and 16 constraints. When Adopt is applied in this scenario, its performance decreases dramatically.

For the class of DMS problems, the authors have proposed two heuristics to speedup the basic Adopt (which has not performed as good as in the graph coloring scenario discussed above). These heuristics have shown good results. One heuristic converts the constraint graph into a deep-first search (DFS) tree which is used as a hierarchy to communicate the values and costs. The authors also suggested to replace this structure with a MLSP (Minimum Depth Spanning) tree. They have shown that the communication structure affect the time of convergence to the optimal solution. Besides, the pre-computation of a best case bound in a distributed fashion was proposed. It was shown that the initial accuracy of this bound affects the convergence in complex scenarios. These heuristics were evaluated in the DMS scenario described above and the total number of cycles felt to less than 15,000. Despite this decrease, in a real application, to exchange 750,000 messages (15,000 cycles times 50 messages per cycle) is still a problem.

Since OptAPO has shown a better performance than Adopt in the graph coloring scenario, we expect a similar performance in the DMS one. Initially we intended to compare the original results of Adopt with the results archived in the scenario proposed above using OptAPO. However, it was not possible due the time complexity associated with the scenario. Next we analyze what happens.

The OptAPO algorithm uses a partial centralization technique (cooperative mediation) based on a centralized search mechanism (B&B) to achieve the optimal result. The performance of OptAPO is closely related to the B&B. Each time an agent decides to mediate in OptAPO (active or passive) it conducts a B&B search in their good_list. As the size of the good_list grows, so does the size of the B&B search space. If we have N variables in the good_list, and M is the domain size, then the size of the B&B search space is M^N in the worst case. Our scenario has 23 variables with an 8-elements domain, which means to search within a 6×10^{20} possibilities. The number of variables involved in the B&B increases dramatically the search space in scenarios with large domains.

During the OptAPO execution, the agents' `good_list` tends to increase as additional links are created due to external conflicts. Through this process, there is a tendency of at least one agent achieving complete centralization and its `good_list` will have all variables. So, there are B&B searches during the OptAPO execution that involve all variables, which means the worst case.

Therefore, since OptAPO would not run for the above scenario, we first reduced the complexity of it. Later we will return to it, via an heuristic approach. Basically, the reduction is achieved by abandoning the PEAV and using an EAV (see Section 3) instead, in which there are less variables (8). Adopt was executed for the same scenario, and the computer configuration was the same. Table 1 shows the runtime (in seconds), the number of exchanged messages, and the number of cycles of OptAPO, a centralized search based on B&B, Adopt, and Adopt with the speedup heuristics (here after we call it Adopt+H).

	OptAPO	B&B	Adopt	Adopt+H
runtime (s)	61133	56641	12529	89
messages	768	-	3370700	15798
cycles	19	-	146566	701

Table 1: Evaluation of DCOP algorithms in the DMS/EAV scenario

Interestingly, OptAPO is worse than the B&B. Let us see why there was a degradation in the OptAPO performance from the graph-coloring scenario to this one. Two heuristics were used to speedup the B&B inside the OptAPO algorithm: the first branch of the search was the current solution; and the search terminates early when the bound is equal to the current optimal local cost and the cost for the agents outside the mediation is 0. These heuristics are important because the quality of the B&B for a specific problem depends on how the branching takes place and which bounding scheme is used. Let us focus on the initial steps of OptAPO execution, where the performance of B&B should, theoretically, be the best due to the `good_list` size. In an over-constraint scenario the agents' `good_list` start with many variables (the `good_list` is first composed by variables' that shares a relation with the agent by a constraint). The primary values of the variables are randomly setup and the current optimal local cost is equal to 0. Due to these characteristics, the speedup proposed for the B&B does not affect the initial phase of the OptAPO algorithm. There is no optimal solution computed yet - the current solution is randomly determined, there are many agents in the `good_list` and no early termination in the search is possible (the bound could not be equal to zero).

For example, when simulating our reduced DMS scenario (EAV), the first search involves all 8 variables, which means a search space of 16 million possibilities. The agent who decides to mediate first (the agent with the highest priority, i.e. the one with more neighbors) has constraints with all other 7 variables. In this case, the initial solution is very poor and better solutions are found only gradually. In summary, when OptAPO is used in this scenario, it has to explore the entire solution space degrading the performance already in the first

steps. As the size of the `good_list` grows, so does the search space of the B&B, increasing the time consumed by OptAPO. Moreover, OptAPO has to run other centralized searches during the further steps of its execution, potentially with more than one including all variables.

The Adopt+H shows the best results due to the speedup heuristics, outperforming all other algorithms. In particular, Adopt without heuristics outperforms OptAPO in terms of runtime. The same arguments used above to justify OptAPO's low performance also apply here. Since Adopt outperforms the centralized B&B already for simple scenarios, and OptAPO was outperformed by B&B here, we expected this to happen. Also, this could be forecasted given the graphs presented in [Mailler and Lesser, 2004], when the authors discuss the performance of Adopt and OptAPO regarding more dense graph (higher number of constraints). Although OptAPO starts with the best performance (in terms of runtime), there is a trend of this not continuing for more dense graphs because OptAPO's performance decreases exponentially while Adopts decreases linearly.

However, and this is very important in real-world scenarios, the number of messages exchanged by OptAPO (as well as the number of cycles necessary to achieve the optimum) is lower than both Adopt and Adopt+H, showing the value of cooperative mediation. Even if Adopt+H outperforms the other algorithms by a large difference in terms of runtime, the number of exchanged messages and cycles taken in this reduced scenario is significantly large.

Assuming that in real-world applications it is normally enough to have a solution close to the optimal, in order to take advantage of the value of the cooperative mediation, we propose to trade the completeness of the B&B search mechanism for the performance of a heuristic search. Our aim is to have the best of the two worlds: OptAPO's performance in terms of number of messages and cycles, and performance in terms of runtime. Therefore, instead of using the B&B algorithm within OptAPO, we use a GA for the search. Of course, this mechanism does not guarantee optimality. However, GAs works well in very large search spaces. Besides, the GA mapping fits this problem well, as demonstrated by the pure-GA based approaches proposed for scheduling and optimization problems [Goldberg, 1989].

In GA, each string is a possible solution having as many genes as variables appearing in the DCOP. We represent each element of the domain as an integer. In order to fit the gene and string mapping, integers were converted to binaries. For example, in our reduced (EAV) DMS scenario there are 8 elements in the domain and 8 variables. Each element in the domain represents a possible meeting's start time x^k . As the agenda has 8 time slots and each meeting consumes one of them, the domain values are 0 to 7. Each element of the domain is represented by 3 binary digits. Thus, the string has 8 binary elements, one for each variable. Considering these, each one representing a meeting E^k , one of the optimal solutions for this scenario (obtained from an Adopt+H run) is: $E^0 = 3$, $E^1 = 0$, $E^2 = 2$, $E^3 = 5$, $E^4 = 7$, $E^5 = 2$, $E^6 = 7$, and $E^7 = 1$. This solution is represented as 01100001010111010111001 in the population.

Each generation has 200 strings. The fitness function is the

same DCOP global objective function discussed in Section 1. The cost of a solution is computed as a sum of the costs of each constraint. Strings are ordered according to the cost and only a set with 40% of the lowest cost solutions are chosen for reproduction. The best 5% of this set stay unchanged in the next generation; the other 35% reproduce by crossover. From this able-to-reproduce population, a small number of genes are mutated with a rate of $\frac{1}{\text{gene_size} \times \text{num_of_genes}}$. In our example, only 0.02% mutates. The solution of the GA is the string with the lowest cost after 200 generations.

Table 2 shows the performance of OptAPO using the GA search mechanism (HeuAPO), Adopt+H, and, for sake of comparison with a centralized search, one based only on GA. The runtime, the total number of cycles, and the number of exchanged messages were measured. Here we can afford to use the more complex DMS scenario (PEAV with 23 variables) described in the beginning of this section. The HeuAPO outperforms Adopt+H in all measured parameters. Besides, HeuAPO reaches the optimal solution in this scenario just as Adopt does.

	HeuAPO	Adopt+H	GA
runtime (s)	978	8268	83
messages	5934	906758	-
cycles	192	13991	-

Table 2: Evaluation of the heuristic algorithm in a DMS scenario, compared to Adopt, and to a centralized search based on GA

5 Conclusions and Further Work

In this paper we analyze the performance of several approaches to solve real-world distributed meeting scheduling problems modelled as distributed constraint satisfaction problems, two well-known scenarios in AI. We compare the performance of the OptAPO and Adopt algorithms. Adopt needs a large number of messages and cycles to converge to a solution, while OptAPO performs worse in terms of runtime, both critical issues in real-world applications. For instance, it was shown that the main assumptions about OptAPO in simple scenarios are not necessarily valid in more complex ones. In our experiment, in terms of runtime, OptAPO was worse than the B&B used to do the internal centralized search, and worse than Adopt, due to the reasons explained in the previous section. However, OptAPO outperforms Adopt in the number of exchanged messages and number of cycles.

Given the performance of Adopt and OptAPO, we have proposed the use of an heuristic search mechanism to replace the B&B in the cooperative mediation. The results obtained are very promising: the heuristic version of OptAPO achieves the best solution with a significant better performance, outperforming Adopt even with the speedup heuristics.

In the future we intend to pursue three directions. First, to run our approach with several different scenarios to check the quality of the solutions. Second, to compare the results of our proposal with other incomplete, heuristic DCOP algorithms, including a version of Adopt which uses an error

threshold thus permitting the algorithm to stop earlier (when the solution is within the threshold). Finally, it would be interesting to investigate which classes of problems are adequate for which type of DCOP algorithm. Liu and Sycara [Liu and Sycara, 1995] discuss the use of partial overlap among subproblems solutions. In their algorithm, *Anchor&Ascend*, an anchor agent is dynamically chosen and conducts a local optimal search in its subproblem. According to the authors, their approach is really effective when the structure of the problem exhibits extreme disparity among the subproblems. Maybe it is the case that OptAPO is also sensible to the structure of the problem, so that not only the complexity of the DMS problem could be affecting OptAPOs performance, but also its structure. Therefore, a similar analysis (algorithm performance vs. problem structure) should be done regarding OptAPO.

References

- [Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co. Inc., 1989.
- [Liu and Sycara, 1995] JyiShane Liu and Katia P. Sycara. Exploiting problem structure for distributed constraint optimization. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 246–254, San Francisco, CA, 1995. MIT Press.
- [Maheswaran *et al.*, 2004] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 310–317, July 2004.
- [Mailler and Lesser, 2004] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 438–445, July 2004.
- [Modi *et al.*, 2003] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. In *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 161–168, 2003.
- [Sen and Durfee, 1991] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling. In *Conference on Organizational Computing Systems*, pages 310–317, September 1991.
- [Yokoo *et al.*, 1998] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.