УДК 004.4

Гороховський С. С., Галковська Л. О.

МОДИФІКАЦІЇ АЛГОРИТМУ GLoSS ДЛЯ РОЗВ'ЯЗАННЯ FDCSP

У цій роботі подано розроблені автором модифікації алгоритму GLoSS, що розв'язують розподілену задачу задоволення обмежень із гнучкими обмеженнями. Наведено доведення повноти та коректності алгоритмів, а також оцінки їх часової складності.

Ключові слова: задача задоволення обмежень із гнучкими обмеженнями, розподілена задача задоволення обмежень із гнучкими обмеженнями, ступінь задоволеності обмежень.

У галузі штучного інтелекту існує клас комбінаторних задач – CSP-задачі (Constraint Satisfaction Problems, CSP), що є потужним засобом для розв'язання практичних задач, що можуть бути спроектовані на множину змінних, пов'язаних між собою обмеженнями. Розв'язком CSP вважають таку комбінацію допустимих значень всіх її змінних, яка задовольняє всім обмеженням [3]. Така модель проте є досить примітивною для того, щоб бути прототипом більшості реальних життєвих задач, оскільки в реальних задачах часто задовільним буває розв'язок, де не всі обмеження задоволено, або не всі задоволено повністю. Ці фактори призвели до того, що з'явився різновид CSP-задач – FCSP (Fuzzy Constraint Satisfaction Problem-FCSP). На відміну від CSP-задач, FCSP-задачі оперують нечіткою логікою та гнучкими обмеженнями, що дає можливість неповного задоволення обмежень у задачі.

1. Зведення FCSP-задачі до CSP

Розглянемо модель FCSP-задач, запропоновану автором статті [4], яку можна вважати базовою FCSP-моделлю. Уявімо, що простір усіх можливих рішень – це U, і він складається з векторів вигляду $(x_1, ..., x_k)$, де k – кількість змінних у задачі, а x-и набувать усіх можливих значень зі своїх доменів. Тепер введемо функцію «ступеня належності» A: $U \rightarrow [0,1]$, яка визначатиме «ступінь належності» кожного вектора $(x_1, ..., x_k)$ до множини розв'язків, або міру, з якою цей розв'язок задовольняє всі обмеження задачі. У звичайних CSP-задачах із жорсткими обмеженнями ця функція має вигляд A: $U \rightarrow \{0,1\}$.

Далі автор вводить так званий поріг α. Цей поріг розділяє всі розв'язки FCSP на допустимі та недопустимі за критерієм: якщо ступінь задоволеності обмежень для цього розв'язку менше ніж α, то такий розв'язок є недопустимим, і навпаки. Таким чином розв'язання FCSP-задачі можна звести до розв'язання CSP-задачі, ввівши поріг задоволеності для розв'язків. Треба зауважити, що залежно від вибору порогового значення α, складність задачі може варіюватися: від поліноміальної до NP [4]. Це буде показано далі на прикладі.

Такий підхід до розв'язання FCSP, тобто зведення її до звичайної CSP, сподобався багатьом, адже він не вимагає розроблення принципово нових алгоритмів для розв'язання цього типу задач, а лише внесення невеликих модифікацій до наявних методів.

Наприклад, автори алгоритму ADOPT спробували застосувати описаний принцип до власного алгоритму в [5]. Вони пропонують розглядати FCSP-задачу як набір CSP-задач, де рівень

задоволеності обмежень *cj* визначається пороговим значенням α, а саме: $\{x: c_j(x) \geq \alpha,\ 0 \leq \alpha \leq 1\}$, інакше кажучи, обмеження *cj* вважають задоволеним, якщо воно задоволене з мірою α, або більше. Тоді відповідне гнучке обмеження з FCSP слід розглядати як: $c_j(x) = sum(\alpha \cdot c\{\alpha\}\{j\}(x))$. Розв'язком FCSP-задачі слід вважати допустимий розв'язок CSP-задачі для найбільшого α.

Залишається запитання, як вибирати значення α, для яких слід розв'язувати CSP-задачі, і скільки їх потрібно. Автори статті дають таку відповідь: треба обмежити значення α зверху та знизу, ввівши α_{LB} та α_{UB}:

$$\alpha_{i+1} \geq \alpha_{LB} = max\{\alpha_k : k \leq i, P_{\alpha_k} solved\} \quad (2)$$
$$\alpha_{i+1} \leq \alpha_{UB} = min\{\alpha_k : k \leq i, P_{\alpha_k} overconstrained\}.$$

Процес вибору значення α визначається алгоритмом, подібним до бінарного пошуку $\alpha_i = (\alpha_{LB} + \alpha_{UB})/2$. Це означає також, що кількість CSP-задач не перевищить $log_2 n$.

2. Алгоритм розв'язання DCSP-задач – GLoSS, адаптований до розв'язання FDCSP-задач

Модифікація № 1 алгоритму GLoSS для розв'язання FDCSP-задач. Найпростішою модифікацією алгоритму GLoSS до розв'язання FDCSP може бути модифікація, що базується на схемі, описаній у [5]. Як було описано в пункті 1, FCSP-задачу можна розглянути як набір CSP-задач, де рівень задоволеності обмежень вигляду $c_j(x)$ визначається пороговим значенням α, а саме: $\{x : c_j(x) \geq \alpha,\ 0 \leq \alpha \leq 1\}$, або, іншими словами, обмеження c_j вважають задоволеним, якщо воно задоволене з мірою α, або більше. Розв'язком FCSP-задачі слід вважати допустимий розв'язок CSP-задачі для найбільшого α.

Також автори статті [5] запропонували алгоритм підбору порогового значення α, адже очевидно, що в процесі розв'язку FCSP-задачі можна буде перебрати лише обмежену кількість значень α та відповідно вирішити лише обмежену кількість CSP-задач, причому очевидно, що цей процес має бути якоюсь мірою оптимізований, оскільки алгоритм вибору значень α впливає на складність розв'язання цільової задачі FCSP. Тож у вищезгаданій статті було запропоновано обирати кожне наступне α методом бінарного пошуку: $\alpha_i = (\alpha_{LB} + \alpha_{UB})/2$.

Застосовуючи цей підхід до алгоритму GLoSS, отримаємо таке: сам алгоритм змін не зазнає, натомість він дістає свого роду обгортку, яка визначає порядок виклику головної процедури. Це означає, що алгоритм у своєму початковому вигляді викликатиметься декілька разів, причому кожного разу для розв'язання наступної CSP-задачі. CSP-задачі формуються за таким принципом:

– встановлюється значення для
$\alpha_i = (\alpha_{LB} + \alpha_{UB})/2$, $\alpha_{LB_0} = 0$, $\alpha_{UB_0} = 1$;

– CSP-задача формулюється таким чином, що кожне з обмежень задачі вважають повністю задоволеним, якщо рівень його задоволеності більше або дорівнює α_i;

– якщо вдається знайти розв'язок CSP-задачі, то нижня границя оновлюється $\alpha_{LB} = \alpha_i$, а верхня границя α_{UB} лишається незмінною;

– якщо розв'язок для CSP-задачі знайти не вдається, оновлюється верхня границя $\alpha_{UB} = \alpha_i$, а нижня α_{LB} лишається незмінною.

Щодо умови закінчення роботи алгоритму, то такою умовою в цій реалізації є точність порогового значення, адже значення границі та самого порогового значення можна оновлювати до нескінченності, бо вони є дійсними числами. Очевидно, що чим більше точність значення α, тим більше ітерацій доведеться зробити.

Теорема 1. Модифікований алгоритм GLoSS є неповним і коректним.

Доведення. Згідно з теоремою 1 зі статті [1] алгоритм GLoSS є повним. Модифікований алгоритм GLoSS використовує оригінальний алгоритм GLoSS для розв'язання DCSP-задач, на які розбивається вихідна FDCSP-задача. Отже, алгоритм гарантує знаходження розв'язку для кожної з підзадач вихідної задачі. Кількість CSP-підзадач є нескінченною, тож у найгіршому випадку пошук може тривати нескінченно, якщо в алгоритм не передати вхідним параметром точність розв'язку, яка і буде критерієм зупинки алгоритму та гарантуватиме його повноту.

Алгоритм на кожній ітерації звужує проміжок допустимих значень, тим самим наближуючись до розв'язку задачі. Це відбувається з допомогою бінарного пошуку, який і гарантує коректність алгоритму. Кінець доведення.

Теорема 2. Для розв'язання FDCSP-задачі з N змінними, у якій потужність кожного з доменів змінних не перевищує M, а потужність домену рівню задоволеності задачі не перевищує K, часова складність модифікованого алгоритму GLoSS є $O(M^N \cdot log_2 K)$. Середня часова складність є $O(M \cdot N \cdot log_2 K)$.

Доведення. Часова складність алгоритму GLoSS згідно з [1] становить $O(M^N)$, а середня часова складність $M \cdot N$. Модифікований алгоритм GLoSS використовує оригінальний алгоритм GLoSS для розв'язання $log_2 K$ DCSP підзадач вихідної задачі

FDCSP. Кількість підзадач log_2K зумовлюється використанням бінарного пошуку в алгоритмі.

Таким чином, складність модифікації GLoSS дорівнює $M^N \cdot log_2K$ для найгіршого випадку, а середня часова складність модифікації GLoSS буде $M \cdot N \cdot log_2K$. Кінець доведення.

Модифікація № 2 алгоритму GLoSS для розв'язання FCSP-задач. Іншим підходом до адаптування алгоритму GLoSS до розв'язання FCSP-задач є модифікація самих компонент для роботи із гнучкими обмеженнями. Цей підхід не вписується в схему, запропоновану авторами [5], і потребує втручання у роботу компонент алгоритму з метою змусити їх працювати з гнучкими обмеженнями.

Отже, алгоритм GLoSS складається із трьох компонент: SBT, iGL (що є модифікацією DisGLS) та AWCS. Перша компонента виконує задачу розширення часткового допустимого розв'язку, допоки всі обмеження лишаються задоволені. Для FCSP цю умову можна описати так: якщо можливо знайти таке присвоєння для чергової змінної, яке задовольняє встановлений рівень задоволеності обмежень, тобто коли всі обмеження, пов'язані із цією змінною, задоволені із рівнем, що більше або дорівнює встановленому пороговому значенню, то таке присвоєння вважається прийнятним, і розширення часткового розв'язку таким значенням є допустимим.

Далі в оригінальному алгоритмі GLoSS при виникненні конфліктних ситуацій, тобто таких, де наступна змінна не може підібрати собі значення, що не конфліктувало би зі значеннями інших змінних, розпочинає свою роботу друга компонента – iGL. У випадку FCSP конфліктною ситуацією є те, що щойно приєднана до часткового розв'язку змінна не може задовольнити всі пов'язані з нею обмеження із заданим рівнем α. У такому випадку необхідно шукати такий частковий розв'язок для цього набору змінних, який би задовольняв заданому рівню α, або якщо це неможливо, знайти найкрайший серед гірших і понизити планку α.

Можна вирішення обох цих проблем доручити другій компоненті iGL, або ж розділити їх між другою та третьою компонентою. Розглянемо другий варіант як модифікацію 2.1 алгоритму GLoSS, а перший – як модифікацію 2.2.

Модифікація 2.1 алгоритму GLoSS для розв'язку FDCSP-задач. У цій модифікації перші дві компоненти гібридного алгоритму GLoSS залишаються практично без змін, розв'язуючи замість FDCSP-задачі конкретну DCSP-задачу, яка утворилася на цьому етапі. Мається на увазі, що перша компонента має чітко встановлений рівень задоволеності α, якому мають відповідати значення всіх обмежень часткового розв'язку FDCSP-задачі, отже, по суті перша компонента розв'язує DCSP-задачу, де всі часткові розв'язки, які ≥ α, є допустимими, а решта – ні. Друга компонента керується таким самим принципом.

FDCSP-задачі, як правило, не можуть бути задоволені із рівнем 1, а GLoSS завжди починатиме свою роботу з припущення, що задача може бути задоволена із найвищим рівнем задоволеності 1. Тож необхідно понизити рівень задоволеності α до максимально можливого для цього часткового розв'язку, якщо ні перша, ні друга компонента не змогли знайти відповідного розв'язку для нього. В цій модифікації цим очевидно має займатися третя компонента AWCS. Зміни, які мають бути внесені для цього в оригінальний алгоритм, відображено в Алг. 1а та Алг. 1б.

В Алг. 1а наведено код процедур check_agent_view оригінального алгоритму AWCS, а в Алг. 1б – його модифікованої версії, де темним кольором виділено відмінності в коді алгоритмів. Оскільки AWCS у цьому розділі розглядається в контексті розв'язку FDCSP-задач, то про сумісність значень агентів можна казати тільки щодо допустимого рівня задоволеності α. Отже, тепер усі значення агентів проходять саме цю перевірку замість перевірки на сумісність, за якої обмеження, що їх зв'язує, повинно мати рівень задоволення 1. Цим самим критерієм модифікований AWCS користується і під час вибору нового значення для агента.

Також відбулися зміни у процедурі backtrack (алг. 2), яка тепер стала відповідальною за пониження планки допустимого рівня задоволеності часткового розв'язку

```
procedure check_agent_view ( )
{
  if curr val is not consistent
    select new value
    if can't find consistent val
      backtrack()
    else
      send ok message to all
    end if
  end if
}
```

Алг. 1а. Псевдокод процедур AWCS

```
1  procedure check_agent_view ( )
2  {
3    if curr val exceeds viol level
4      select new value
5      if can't find acceptable val
6        backtrack()
7      else
8        send ok message to all
9      end if
10   end if
11 }
```

Алг. 1б. Псевдокод процедур модифікованого AWCS

задачі, якщо алгоритм виявив, що всі можливі розв'язки були перебрані, і жоден з них не виявився задовільним. Це відбувається в момент, коли оригінальний AWCS мав би повернути повідомлення про те, що розв'язку для цієї задачі не існує. Пониження рівня задоволеності відбувається дискретно, тобто на якусь константну задану величину. Причому, якщо ця величина буде обрана досить великою, то це може зменшити точність розв'язку. Якщо ж вона навпаки буде обрана дуже маленькою, то може суттєво збільшити час розв'язання задачі.

```
procedure backtrack ( )
{
  add state to nogood store
  for all agents with higher prior

    if not consist with our agent
      send nogood message
    end if

   prior ← max(all agents prior)+1
  select new value with min number of
constraint violations

  send ok message to all
}
```

Алг. 2а. Псевдокод процедур AWCS

```
1   procedure backtrack ( )
2   {
3     add state to nogood store
4     for all agents with > prior
5       send nogood message
6     if no agents with higher prior
7       increase violation level
8     end if
9     prior ← max(all agents prior)+1
10    select new value with min number
11  of   constraint violations
12
13    send ok message to all
14  }
```

Алг. 2б. Псевдокод процедур модифікованого AWCS

Інші процедури залишилися без змін.

Теорема 3. Модифікований алгоритм AWCS є повним і коректним.

Доведення. Алгоритм AWCS є алгоритмом конструктивного пошуку, що гарантує його повноту, тобто гарантує знаходження допустимого часткового розв'язку, якщо такий існує, або завершення роботи алгоритму за браком рішення.

Модифікована версія алгоритму має дві основні відмінності:

1. Рівень задоволеності часткового розв'язку в оригінальному AWCS є константою та становить 1. Для модифікованого AWCS цей рівень представлений α, і допустимими вважають усі часткові розв'язки, рівень задоволеності яких ≥ α. Причому доки алгоритм не знайде допустимий розв'язок, або не доведе, що його не існує для заданого α, α не змінюється.

2. Якщо алгоритм перебрав усі можливі варіанти і не зміг знайти допустимого розв'язку, він зупиняє свою роботу. Модифікована версія в цей момент понижує рівень задоволеності обмежень α на деяку константу і по суті починає процес розв'язання заново. Кількість таких ітерацій скінченна, і мінімальним значенням α є 0.

Наведеними двома пунктами гарантується, що алгоритм завжди завершує свою роботу і що отриманий розв'язок матиме максимально можливий рівень задоволеності α. Кінець доведення.

Теорема 4. Для розв'язання FDCSP-задачі з N змінними, у якій потужність кожного з доменів змінних не перевищує M, часова складність алгоритму GLoSS є $O(M^N)$. Середня часова складність є $O(M^N)$.

Доведення. Згідно з теоремою 2 зі статті [1] складність алгоритму GLoSS є $O(M \cdot N + C + M^N)$ або $O(M^N)$. Тут останній доданок відображає часову складність компоненти AWCS.

В оригінальному алгоритмі GLoSS, за теоретичними оцінками, компонента AWCS бере участь у розв'язанні задачі не більше ніж у 3 % випадків, тому середня часова складність GLoSS становить $M \cdot N + C$, або просто $M \cdot N$. У модифікації 2.1 алгоритму GLoSS компонента AWCS відіграє ключову роль і викликатиметься відповідно практично у 100 % випадків.

Складність модифікованого AWCS є $C_2 \cdot M^N$, де C_2 позначає максимальну кількість разів, на яку зменшувався рівень задоволеності α від 1 до 0. Цю константу можна опустити, і в такому разі складність модифікованого AWCS визначатиметься як M^N. Тож у найгіршому випадку складність всього гібрида буде знов $O(M \cdot N + C + M^N)$ або $O(M^N)$.

Середня часова складність модифікованого AWCS є також M^N. Оскільки він викликатиметься практично у 100 % випадків на кожній ітерації GLoSS, то середню часову складність гібрида можна оцінити як $O(M^N)$. Кінець доведення.

Модифікація 2.2 алгоритму GLoSS для розв'язку FCSP-задач. В ідеалі, AWCS-компонента гібрида не повинна викликатися взагалі, або викликатися достатньо рідко для того, щоб показувати найкращі результати з погляду продуктивності. В такому разі логічно перекласти відповідальність за пониження рівня задоволеності обмежень α на другу компоненту iGL. Втім, третя компонента все одно має підстраховувати локальний алгоритм на випадок невдачі, адже він є неповним.

В Алг. 4а та 4б наведено код процедур модифікації 2.2 алгоритму GLoSS, які зазнали змін щодо оригіналу. Модифікований алгоритм також має одну нову процедуру iGL_has_become_worse (Алг. 3). Оскільки новий алгоритм допускає

```
procedure iGL_main
{
 do
  when active
   evaluate state
   if penalty message received
    responde_to_message()
   else
    if curr val is consistent
     reset inc penalties
    else
     if sender prior < cur prior
      resolve_conflict()
     else
      send(id_val,null) back
     end if
    end if
   end if
   return to inactive state
  until terminate
}
```

Алг. 3а. Головний цикл алгоритму iGL

```
1  procedure iGL_main
2  {
3   do
4    when active
5     evaluate state
6     if penalty message received
7      if curVal == val in sender view
8        responde_to_message()
9      end if
11    else
12     if not iGL_has_become_worse()
13       reset inc penalties
14     else
15       if sender prior < cur prior
16         resolve_conflict()
17       else
18         send(id_val,null) back
19       end if
20     end if
21    end if
22    return to inactive state
23   until terminate
24 }
```

Алг. 3б. Головний цикл модифікованого алгоритму iGL

пониження планки α до невизначеного рівня, він повинен мати механізм розрізнення ситуацій, коли α має бути знижена у зв'язку з тим, що розв'язку для більшого α дійсно не існує, та ситуацій, коли для зменшення α немає причин, оскільки алгоритм просто потрапив у локальний мінімум, і ми точно знаємо, що існує і кращий розв'язок. Реалізація такого механізму полягає у перевірці того, чи поточний розв'язок має менший рівень задоволеності, ніж попередній, і якщо так – отже існує як мінімум одне краще рішення, тому необхідно завадити алгоритму понизити планку α.

Ці зміни також відображені в головній процедурі алгоритму iGL_main (Алг. 4а та Алг. 4б). Окрім цього, процедура містить ще одну зміну, яка стосується насамперед вирішення проблеми синхронізації між агентами. Оскільки дві з трьох компонент алгоритму GLoSS є асинхронними, порядок і час доставки повідомлень до агентів не може гарантуватися, отже, інколи отримуючи penalty повідомлення від одного із сусідів, агент не може бути впевнений, що його сусід має такий самий agent_view, як і в нього, а тому може йому і не потрібно накладати на себе ніякі штрафи, а просто розіслати своє поточне значення.

```
procedure iGL_has_become_worse                1
{                                             2
  if cur violation level <                    3
    violationLevel(curVal,senderVal)          4
    return true;                              5
  else                                        6
    return false;                             7
 end if                                       8
}                                             9
```

Алг. 4. Процедура has become worse модифікованого алгоритму iGL

Теорема 5. Модифікований алгоритм 2.2 є повним і коректним.

Доведення. Алгоритм GLoSS є повним алгоритмом, що доводиться в теоремі 1 у статті [1]. Модифікація GLoSS 2.2 також є повною, адже модифікований алгоритм, як і оригінальний, складається з трьох компонент, серед яких перша компонента є повною, друга, iGL, – неповною, але закінчення її роботи гарантується обмеженою кількістю кроків, яка надається компоненті для розв'язання підзадачі, після чого підзадача передається третій компоненті – AWCS, яка є повною і гарантує завершення розв'язку. Повноту третьої модифікованої компоненти AWCS доведено вище в теоремі 3.

Коректність модифікованого алгоритму GLoSS можна довести через коректність його компонент. Коректність першої компоненти, яка не зазнала ніяких модифікацій, доведено тут [6]. Коректність третьої компоненти доведено вище в теоремі 3. Коректність другої компоненти, а саме модифікованого алгоритму iGL, можна довести через коректність оригінального алгоритму iGL. Єдиною їхньою відмінністю є підміна умови в головній процедурі алгоритму (алг. 4а, 4б, рядок 12), яка визначає необхідність вирішення конфлікту поточним агентом. В оригінальному алгоритмі такою умовою є перевірка допустимості поточного розв'язку, а у модифікованому алгоритмі – виклик процедури iGL_has_become_worse. Якщо рівень задоволеності часткового розв'язку погіршився – це прирівнюється до «недопустимості часткового розв'язку», тоді як будь-який інший варіант можна розглядати як допустимий. Таким чином паралель між двома

алгоритмами проведена, ідентичність їхньої поведінки показана, а отже повнота модифікованого алгоритму доведена. Кінець доведення.

Теорема 6. Для розв'язання FDCSP-задачі з N змінними, у якій потужність кожного з доменів змінних не перевищує M, часова складність алгоритму GLoSS є $O(M^N)$. Середня часова складність є $O(M \cdot N)$.

Доведення. Згідно з теоремою 2 зі статті [1] складність алгоритму GLoSS є $M \cdot N + \ + C + M^N$ або $O(M^N)$. Тут перший доданок відображає часову складність компоненти SBT, другий – iGL, а третій – AWCS.

У модифікованому алгоритмі GLoSS складність першої компоненти буде такою самою, як і в оригінальному, оскільки ця компонента змін не зазнає, тож вона дорівнюватиме $M \cdot N$. Складність другої компоненти, як і в оригінальному алгоритмі, можна оцінити як константу C, оскільки компоненті відводиться тільки обмежена кількість кроків, у зв'язку з її неповнотою. Третя ж компонента, як було показано в доведенні теоремі 4, має складність $C_2 \cdot M^N$, або просто M^N. Таким чином складність модифікованого GLoSS можна оцінити як $M \cdot N + C + M^N$, або просто M^N.

В оригінальному алгоритмі GLoSS, за теоретичними оцінками, компонента AWCS бере участь у розв'язку задачі не більше ніж у 3 % випадків, тому середня часова складність GLoSS становить $M \cdot N + C$, або просто $M \cdot N$. Це справедливо і для цієї модифікації алгоритму GLoSS. Кінець доведення.

Висновки

FDCSP-задачі є класом задач, похідним від CSP. Вони з'явилися внаслідок виникнення необхідності формулювати обмеження в CSP-задачах у більш гнучкій формі. Проте потужність арсеналу методів розв'язання FCSP-задач значно поступається кількості методів розв'язання CSP-задач. Причому важко сказати, чому так сталося: чи причина полягає в тому, що розробка принципово нових методів для розв'язання FCSP-задач виявилася надто важким завданням, чи просто неперспективним. Враховуючи той факт, що одночасно із виникненням самого поняття FCSP були сформульовані й основні принципи зведення FCSP-задач до CSP, і як наслідок можливість використання всього арсеналу методів розв'язання CSP задач до FCSP, мабуть усе-таки причина полягає в іншому. Хоча деякі спроби звернути з цього шляху все ж були здійснені, хоча не можна сказати, що вони були успішними. Наприклад, ця стаття розказує про методи розв'язання FCSP з допомогою нейронних мереж [2].

Ця стаття продемонструвала черговий успішний приклад застосування методів зведення FCSP-задач до CSP на прикладі GLoSS, який початково розроблявся як CSP-алгоритм, але був успішно адаптований до розв'язання FCSP, причому у трьох модифікаціях.

Перша з розглянутих модифікацій наочно демонструє схему зведення FCSP до CSP, запропоновану в [4]. Такий спосіб адаптування алгоритму до розв'язання іншого типу задач є найпростішим, адже він не вимагає внесення змін до самого алгоритму, а тільки розбиття початкової FCSP-задачі на CSP-підзадачі, які потім одна за одною подаються на вхід алгоритму. Проте як показала оцінка продуктивності всіх модифікацій, цей алгоритм посів друге місце серед трьох.

Дві інші модифікації, наведені в статті, передбачають внесення змін в ті чи інші компоненти оригінального гібридного алгоритму GLoSS, що відбувається вже не за відпрацьованими іншими авторами схемами, а в творчому порядку. Такий експеримент дав змогу розробити модифікацію алгоритму GLoSS 2.2, яка за оцінкою ефективності обійшла навіть першу модифікацію, розроблену за перевіреною схемою. Єдиним мінусом цієї модифікації можна вважати те, що вона поки не узагальнена до абстрактної моделі, яка може бути застосована до адаптування інших алгоритмів схожим чином, проте створення такої моделі є основною ідеєю подальших досліджень з цієї теми.

Список літератури

1. Галковская Л. Гибридный алгоритм решения задачи удовлетворения ограничений / Л. Галковская, М. Глибовец, С. Гороховський // Информационные технологии. Управляющие системы и машины. – 2012. – Ноябрь – декабрь. – Вып. № 6. – С. 72–87.
2. Галковская Л. Застосування нейронних мереж Хопфілда для розв'язання CSP задач / Л. Галковская // Наукові записки НаУКМА. – 2015. – Т. 177 : Комп'ютерні науки. – С. 16–24.
3. Brito I. Synchronous, Asynchronous and Hybrid Algorithms for DisCSP. Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP-2004) / I. Brito // Lecture Notes in Computer Science. – 2004. – Vol. 3258, January. – P. 791.
4. Hermann M. On the Complexity of Fuzzy Boolean Constraint Satisfaction Problems With Applications to Intelligent Digital Photography, Circuits, Communications and Systems (PACCS) / M. Hermann, F. Richoux // Proceedings of the Third Pacific-Asia Conference. – 2011. – July. – P. 1–4.

5. Nguyen X.-T. On Solving Distributed Fuzzy Constraint Satisfaction Problems with Agents / X.-T. Nguyen, R. Kowalczyk // Proceedings of the International Conference on Intelligent Agent Technology. – 2007. – P. 387–390.

6. Yokoo M. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Trans / M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara // Knowledge and Data Engineering. – 1998. – P. 673–685.

S. Gorokhovskyy, L.Galkovska

MODIFICATIONS OF THE ALGORITHM GLOSS FOR SOLVING FDCSP

This work introduces three new algorithms for solving Fuzzy Distributed Constraint Satisfaction Problem (FDCSP). They all are modifications of the author's algorithm for solving DCSP problems – GLoSS. Modifications were built using the method proposed by the authors of ADOPT algorithm for converting CSP algorithms for solving CSP problems to the ones that solve FCSP. Given that GLoSS is a hybrid method and consists of three components, its three different modifications were created. The first component – SBT – needed no modification but required some corrections of the exit condition. The same can be said about the second component iGL. The only big change that has been made in it is that the modified version is attempting to find the solution which satisfies all the constraints with some given satisfaction level, which is not necessarily 1. The third component – AWCS – experienced the biggest changes between all three GLoSS components. Now its exit condition has been changed so that it restarts with the new configuration instead of returning "no solution" answer.

All these algorithms, specifically those parts that have changed from the original version, are supported with the pseudo-code. Also their completeness and correctness is proved by the corresponding theorems. Besides that, the article contains complexity estimations for the algorithms and their brief comparison, which may help to choose the right algorithm depending on the type of the problem to solve.

Keywords: Fuzzy Constraint Satisfaction Problem, Fuzzy Distributed Constraint Satisfaction Problem, satisfaction degree.

Матеріал надійшов 23.10.2016