Список літератури

1. Bengio Y. Gradient–based optimization of hyper–parameters / Y. Bengio // Neural Computation. – 2000. – Vol. 12(8).
2. Chapelle O. Model selection for support vector machines / O. Chapelle, V. Vapnik // Advances in Neural Information Processing Systems. – 1999.
3. Cortes C. Support vector network / C. Cortes, V. Vapnik // Machine learning. – 20: 1–25. – 1995.
4. Cristianini N. Dynamically adapting kernels in support vector machines / N. Cristianini, C. Campbell, J. Shawe-Taylor // Advances in Neural Information Processing Sustems. – 1999.
5. Freund Y. Large margin classification using the perceptron algorithm / Y. Freund, R. E. Schapire // Computational Learning Theory. – 1998. – P. 209–217.
6. Platt J. Probabilities for support vector machines / J. Platt // Advances in Large Margin Classifiers / ed. A. Smola, P. Bartlett, B. Schölkopf, D. Schurmans. – Cambridge, MA : MIT Press, 2000.
7. Rätsch G. Soft margins for AdaBoost / G. Rätsch, T. Onoda, K.-R. Müller // Machine Learning. – 2001. – 42(3). – P. 287–320.

O. Galkin

AUTOMATIC SELECTION OF THE PARAMETERS OF THE SUPPORT VECTOR MACHINES KERNEL

Methodology of the automatic selection of the support vector machines kernel is investigated. Gradient descent approach is used for minimizing test error estimates. A smooth approximation of the test error is obtained by estimating posterior probabilities. An algorithm is proposed and implemented where a gradient step is the direction of the gradient in the parameters space. Experiments have been carried out to assess the performance and feasibility of the proposed method.

Keywords: kernel, gradient descent approach, smooth approximation of the test error, validation set.

Матеріал надійшов 22.10.2012

УДК: 004.4

Галковська Л. О.

АЛГОРИТМИ РОЗВ'ЯЗАННЯ РОЗПОДІЛЕНОЇ ЗАДАЧІ ЗАДОВОЛЕННЯ ОБМЕЖЕНЬ

У даній роботі представлено розроблену автором класифікацію алгоритмів розв'язання розподіленої задачі задоволення обмежень. Наведено приклади представників кожного з класів, а також їх порівняльна характеристика та рекомендації щодо їх застосування для розв'язання конкретних DCSP задач.

Ключові слова: задача задоволення обмежень, розподілена задача задоволення обмежень, алгоритми локального пошуку, алгоритми конструктивного пошуку, гібридні алгоритми.

Багато задач оптимізації та пошуку можуть бути формалізовані як задачі задоволення обмежень (Constraint Satisfation Problems-CSP). Прикладом такої задачі може слугувати відома задача розфарбування графа. З поширенням концепції розподілених обчислень сформувався підклас розподілених CSP задач (Distributed Constraint Satisfaction Problem-DCSP). У багатьох випадках для їх розв'язання залучають групу незалежних виконавців (наприклад, агентів). Кожен агент оперує частиною задачі (частіше за все опікується однією змінною з задачі або однією підзадачею) і обмінюється інформацією з іншими виконавцями, з якими він пов'язаний певними умовами

чи обмеженнями. Така концепція позитивно впливає на швидкість знаходження розв'язку і забезпечує кращу надійність.

Задачу задоволення обмежень можна представити у вигляді трійки (X,D,C), де $X = \{x_1,\ldots,x_n\}$ – множина з n змінних; $D = \{D(x_1),\ldots,D(x_n)\}$ – множина відповідних скінченних доменів цих змінних, тобто множин допустимих значень; C – множина обмежень, накладених на ці змінні. Розв'язком CSP вважається така комбінація допустимих значень всіх її змінних, яка задовольняє всі обмеження [4, с. 791].

Для уточнення постановки задачі у розподіленому варіанті CSP, будемо вважати, що виконавців представляють агенти[1]. Визначимо спілкування між агентами як повідомлення [17, с. 88]. Агент може надсилати повідомлення іншим агентам, якщо він знає їх адреси. Затримка доставки повідомлень скінченна і випадкова. Повідомлення отримуються агентом у порядку їх надсилання.

Тоді формальне визначення DisCSP можна представити як п'ятірку (X,D,C,A,ϕ), де X, D визначені як і в CSP; C – множина обмежень, що накладаються на змінні з X; $A = \{1,\ldots,p\}$ – множина агентів; ϕ: X → A функція, що зіставляє кожному агенту змінні з множини X. Кожен агент може оперувати кількома змінними, але кожна змінна має належати тільки одному агенту. У зв'язку із цим множина C поділяється на дві частини: множина внутрішніх обмежень агента $C_{intra} = \{c_{ij} \mid \phi(x_i) = \phi(x_j)\}$, тобто множина обмежень, накладених на змінні одного агента, та множина обмежень, накладених на змінні різних агентів $C_{inter} = \{ c_{ij} \mid \phi(x_i) \neq \phi(x_j)\}$ [4, с. 791].

Можна довести, що кожна CSP зводиться до бінарного випадку. Тоді функція ϕ зіставляє кожному агенту лише одну змінну, і клас C_{intra} буде порожнім (всі обмеження бінарні, а кожен агент оперує тільки однією змінною).

Введемо також такі позначення: belongs(x_j, i) – змінна x_j належить агенту i; known(c_m, k) – агент k знає про обмеження c_m. Тоді DisCSP будемо вважати розв'язаною у випадку, якщо виконуються такі умови [17, с. 88]:

- $\forall i\ \forall x_j$ belongs(x_j, i) d_j: $x_j = d_j$, тобто всім змінним всіх агентів присвоєно якесь значення з їх доменів.
- $\forall k$, $\forall c_m$ known(c_m, k), c_m виконується для значення d_j, присвоєного змінній x_j.

[1] Агенти – в інтелектуальних системах, деякі сутності, що спостерігають за навколишнім середовищем і діють у ньому, при цьому їхня поведінка раціональна в тому розумінні, що вони здатні до розуміння і їхні дії завжди спрямовані на досягнення якої-небудь мети.

1. Алгоритми розв'язання розподілених задач задоволення обмежень

Алгоритми розв'язання DCSP почали розвиватися в кінці минулого тисячоліття і їх число зростало з кожним роком. Став розвиватися напрям «поліпшення існуючих методів» різноманітними варіаціями відомих алгоритмів. Причому два напрями розв'язання DCSP, конструктивний та локальний, розвивалися паралельно.

Але швидко базові ідеї щодо покращення алгоритмів також були вичерпані. Наразі увагу науковців зосереджено на: 1) розробці спеціальних алгоритмів для конкретних типів задач; 2) об'єднанні існуючих алгоритмів у гібриди, що наслідують переваги одразу кількох методів.

1.1. Алгоритми конструктивного пошуку[2]

Основна ідея конструктивних методів полягає в покроковому розширенні часткового допустимого розв'язку до повного. Якщо частковий розв'язок виявився недопустимим, то відбувається бектрекінг, і попередній розв'язок поширюється в альтернативному напрямі. Той факт, що пошук відбувається у просторі часткових розв'язків, є відмінною рисою цих алгоритмів.

Визнання часткового розв'язку недопустимим позбавляє необхідності розглядати все його піддерево (рис. 1). З іншого боку, метод покрокового розширення допустимого часткового розв'язку суттєво обмежує простір пошуку в кожен момент часу. Невдале початкове рішення може спрямувати пошук у неправильному напрямі, суттєво збільшуючи час роботи алгоритму. В найгіршому випадку це повний перебір

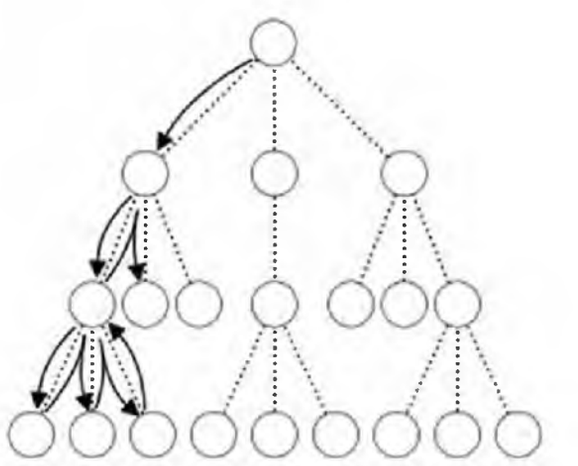

Рис. 1. Загальна схема роботи конструктивного алгоритму [13, с. 29]

[2] У наукових статтях англійською мовою різних авторів термін «конструктивний» позначається різними словами, наприклад: exhaustive, global, systematic, constructive, refinement.

```
constructive-search (a)
{
  L ← 0
  while a is not solution
  {
    if a is consistent
      a ← extend (a, L)

    else if (L ≠ 0)
      a ← backtrack (a, L)

    else return no solution
  }
  return a
}
```

Алг. 1. Загальний алгоритм конструктивного пошуку

мані. Ця множина гарантує повноту методів, що на ній виконуються, тобто вона гарантує, що пошук буде відбуватися по всій множині можливих рішень. Якщо допустимий повний розв'язок у результаті виконання алгоритму не буде знайдено – значить його не існує.

Extend(α, L) – функція, що генерує можливі розширення даного часткового розв'язку α; повертає одне із них, а решту додає до L. Backtrack(α, L) – вибирає та повертає одне з незадіяних досі часткових рішень із множини L [4, с. 791].

У таблиці 1 показано систематизований матеріал по основних існуючих наразі алгоритмах конструктивного пошуку для задач DisCSP.

Усі алгоритми конструктивного пошуку поділяються на дві великі групи за методом синхронізації між агентами: синхронні та асинхронні.

Таблиця 1. **Основні алгоритми конструктивного пошуку для DisCSP**

Назва алгоритму	Короткий опис	Ефективність	Рік
Synchronous Backtracking (SBT) [18, с. 614]	Послідовний; низький рівень взаємодії між агентами.	Добре працює для задачі про n ферзів	1992
Asynchronous Backtracking (ABT) [18, с. 616]	Паралельний; високий рівень взаємодії між агентами.	У 1,5–2 рази швидше від SBT на coarse-grained задачах.	1992
Synchronous Conflict-Based Backjumping (SCBJ) [20]	Покращена версія SBT, де бектрекінг відбувається до конфліктного, а не попереднього агента.	У 2 рази кращий за SBT. Один з найкращих алгоритмів для розв'язку задачі про n ферзів.	2003
Asynchronous Weak-commitment search (AWCS) [17]	Використовує евристичну функцію для упорядкування змінних, запам'ятовує погані рішення.	В 3–10 разів швидший від ABT (за оцінкою авторів алгоритму [17, с. 89]).	1994
Asynchronous aggregation search (AAS) [16, с. 25]	Модифікована версія ABT, тільки змінним замість одного значення присвоюються проміжки значень.	Ефективніший від ABT.	2000
Maintaining Asynchronously Consistencies (MAC) [16, с. 35]	Кожен агент замість стандартного agent view будує ієрархію проміжків-значень сусідів.	В 10 разів ефективніший від AAS (за результатами експериментів авторів алгоритму [16, с. 50]).	2001

з експоненційним часом виконання. Втім такий підхід дає гарантію того, що алгоритм завжди знаходить розв'язок задачі, якщо він існує, і може довести його відсутність, якщо задача не має розв'язку; тож усі алгоритми конструктивного пошуку є повними[3].

Конструктивні методи можуть бути описані загальним алгоритмом (алг. 1):

L – множина всіх допустимих та недопустимих часткових розв'язків, які можуть бути отри-

Синхронні алгоритми організовано таким чином, що під час їх виконання в кожен момент часу активним є тільки один агент, який оперує частковим розв'язком задачі, а після завершення своєї роботи пересилає частковий розв'язок наступному агенту для обробки, або ж завершує роботу у зв'язку зі знаходженням повного розв'язку задачі. Такий підхід сам по собі не використовує переваг розподіленості, адже агенти не мають змоги працювати одночасно і незалежно одне від одного. Асинхронні методи ґрунтуються на тому, що всі учасники процесу розв'язку

[3] У наукових статтях англійською мовою термін «повний» позначається словом complete.

мають змогу працювати одночасно над своїми задачами, тобто, на відміну від синхронних алгоритмів, ці методи використовують максимально ефективно доступні їм ресурси та час замість того, щоб витрачати його на очікування.

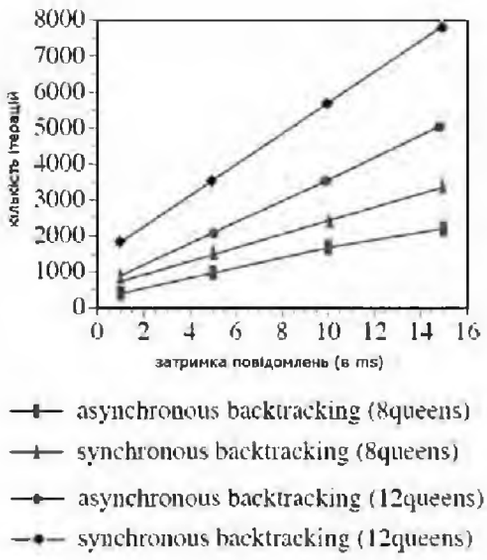

Рис. 2a. Кількість ітерацій алгоритмів SBT та ABT в залежності від величини затримки повідомлень

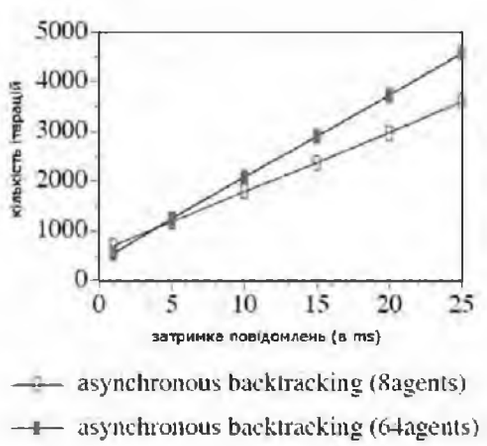

Рис. 2b. Кількість ітерацій алгоритму ABT в залежності від величини затримки повідомлень для задачі з 8 ферзями, де кожен агент оперує одним ферзем; та 64 ферзями, де кожен агент оперує 8 ферзями

Наприклад, у статті [18] наводяться результати експериментального дослідження щодо порівняння ефективності SBT та ABT. Експерименти проводилися на задачі про n ферзів n для випадків n = 8 та n = 12. Результати експерименту було оцінено в кількостях кроків, які знадобилися алгоритмам для знаходження розв'язку, і представлено на рис. 2*a*. Для глибшого аналізу автори моделювали штучну затримку повідомлень в системі (в реальних агентних системах, особливо великих, це нерідко трапляється), варіюючи її розмір від 0 до 16 мілісекунд. Як бачимо на графіку, в усіх випадках асинхронна версія показала кращі результати за синхронну у 1,5–2 рази.

Графік також показує, що час, потрібний алгоритму для знаходження розв'язку, зростає лінійно із зростанням затримки у доставці повідомлень агентам, тобто для класичного бектрекінгу його ефективність залежить від швидкості та якості роботи каналу обміну повідомленнями. Для того, щоб використати ABT з максимально можливою ефективністю, треба подати йому на вхід задачу у правильному вигляді. Це означає, що бектрекінг упорається набагато краще із задачею з 64 ферзями, якщо розподілити їх не по одному ферзю на агента, а задіяти вісім агентів, кожен з яких відповідатиме одразу за 8 ферзів. У цьому випадку кількість повідомлень між агентами зменшиться в рази, і агенти зможуть працювати більш незалежно одне від одного. Це добре видно на рис. 2*b*. Результат роботи ABT на 64 ферзях, розподілених по 8 на агента, навіть кращий, ніж на 16 ферзях, де кожен агент відповідає тільки за одного ферзя [18, с.620].

Для синхронного методу варіант розподілу ферзів по агентах незастосовний, бо це суперечить самій сутності алгоритму, тож він при такому розкладі програє ABT у швидкодії ще в кілька разів. Втім це все ж таки не означає, що всі синхронні методи заздалегідь можна вважати гіршими за асинхронні, і це добре висвітлено авторами статті [20], які у 2003 році представили світу алгоритм SCBJ. Вони розробили власну систему оцінки продуктивності алгоритмів, заснованих на бектрекінгу, яка частково запозичена у [14]. Для оцінки ефективності алгоритмів вони використовують згенеровані випадковим чином CSP, що характеризуються двома величинами: p1 – ймовірність наявності обмеження між будь-якою парою змінних в задачі, p2 – ймовірність конфлікту між значеннями, присвоєними двом змінним, зв'язаним обмеженням. Для експериментів використано задачі з 10 змінних, де кожна змінна мала домен з 10 можливих значень, при цьому параметр p1 був встановлений в 0,7, а значення параметру p2 варіювалося між 0,1 та 0,9. Згенеровані таким чином задачі оцінювалися за двома параметрами: за кількістю перевірок виконуваності обмежень, накладених на змінні, та за кількістю надісланих за час розв'язання повідомлень.

Наведена схема використана авторами статті для порівняння ефективності SBT, ABT та SCBJ і показана на рис. 3 у вигляді двох графіків, де графік а) відображає кількість перевірок на задоволення обмежень, а графік б) – загальну кількість повідомлених, надісланих агентами під час роботи алгоритму.

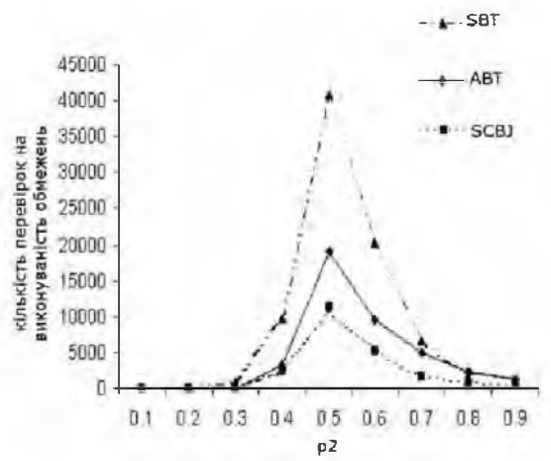

Рис. 3a. Кількість перевірок на виконуваність обмежень в алгоритмах SBT, ABT та SCBJ для задач із різним ступенем щільності обмежень

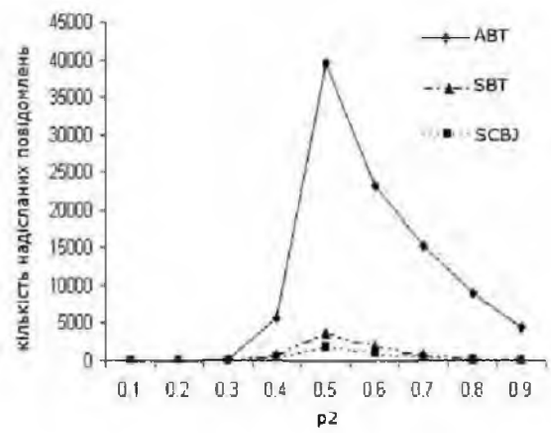

Рис. 3b. Кількість надісланих повідомлень за час роботи алгоритмів SBT, ABT та SCBJ для задач із різним ступенем щільності обмежень

На малюнку бачимо, що асинхронний метод при такому розкладі показує себе не з найкращого боку. Автори статті наголошують на тому, що ABT довгий час вважався гідною і більш ефективною заміною SBT через те, що надмірне завантаження каналу повідомлень нібито мало компенсуватися меншими обчислювальними витратами (зокрема тим, що більшість обчислень виконувалася паралельно). Втім даний графік, на їхню думку, показує те, що насправді все не так просто, і видно, що SBT з його великою кількістю перевірок на виконуваність обмежень і дуже малою кількістю повідомлень у певні моменти показує навіть кращі результати від ABT з його середнім рівнем перевірок і дуже високим рівнем завантаженості каналу повідомлень. Очевидним є й те, що SCBJ в цих експериментах переверщив за продуктивністю обох своїх конкурентів, тобто як ABT, так і SBT (його графік всюди є найменш піковим).

Наступним алгоритмом у змаганні за ефективність виступає AWCS, який є прямим нащадком ABT з тією відмінністю, що він вміє запам'ятовувати невдалі комбінації значень сусідів, щоб у разі наступного потрапляння у заздалегідь програшну ситуацію одразу відмовлятися від руху в цьому напрямку. Також цей алгоритм відрізняється тим, що в ньому розподіл пріоритетів між агентами, характерний для всіх алгоритмів, які базуються на бектрекінгу, є динамічним, тобто вони можуть мінятися ролями в процесі роботи алгоритму. Цей представник сімейства конструктивних алгоритмів нарешті довів, що асинхронні методи все-таки є гідним суперником синхронним версіям і показуватимуть результати в 10 разів кращі і навіть вирішуватимуть задачі, які інші конструктивні алгоритми розв'язували б майже нескінченну кількість часу (тобто на розв'язок їм знадобилася б кількість часу, що не вписується в межі розумного). Результати порівняння алгоритму AWCS із ABT на задачі про n ферзів наведено у таблиці 2, де перший стовпчик вказує кількість ферзів, другий відображає відсоток задач, які алгоритм зміг розв'язати (кількість кроків, доступних алгоритмам, була обмежена, тому в разі, якщо алгоритм не вкладався у цю кількість, задача вважалася нерозв'язаною), і, нарешті, третій стовпчик відображає середню кількість кроків, яка знадобилася алгоритму для розв'язку.

Таблиця 2. **Відсоток задач, розв'язаних алгоритмами ABT та AWCS залежно від кількості ферзів у задачі**

n	ABT		AWCS	
	% розв'язаних задач	к-ть кроків	% розв'язаних задач	к-ть кроків
10	100	105,4	100	41,5
50	50	662,7	100	59,1
100	14	931,4	100	50,8
1000	0	--	100	29,6

Алгоритм AWCS має лише один суттєвий недолік. Кількість пам'яті, доступної для зберігання невдалих рішень, є взагалі-то необмеженою. Втім у реальних умовах вона все-таки має бути обмеженою, але введення таких обмежень призводить до того, що алгоритм перестає бути повним, тобто в такому разі його завершуваність не гарантується.

Останні два представники конструктивного пошуку з таблиці є представниками зовсім іншого напрямку в DCSP, ніж розглядалося

досі. По суті, вони започаткували лінійку розподілених алгоритмів для розв'язку оптимізаційного класу задач COP (Constraint Optimization Problem), які характеризуються підвищеною складністю і їх характерною рисою є неможливість знаходження розв'язку через надмірну кількість обмежень, окремі з них суперечать одне одному, тож для їх задоволення доводиться шукати якісь компроміси. Дані алгоритми, на відміну від усіх попередніх, працюють із проміжками значень, а не одиничними значеннями. З таблиці видно оцінку їх ефективності на задачах DCSP. До речі, вони, судячи з усього, послужили основою для одного з найкращих відомих алгоритмів конструктивного пошуку ADOPT, який не ввійшов до наведеного переліку через те, що він у більшій мірі відноситься до розв'язку оптимізаційних задач.

1.2. Алгоритми локального пошуку[4]

Локальний пошук – основний метод розв'язання особливо складних задач із класу задач пошуку CSP. Крім того, найчастіше він є єдиним можливим способом розв'язання задач, де конструктивні методи безсилі через надто великий розмір простору можливих рішень.

Ідея локального пошуку полягає у тому, щоб почати із довільно згенерованого розв'язку проблеми і, якщо він виявиться недопустимим, покроково його поліпшувати шляхом зменшення кількості незадоволених обмежень. Алгоритми локального пошуку відрізняються методами знаходження цього покращення [16–20].

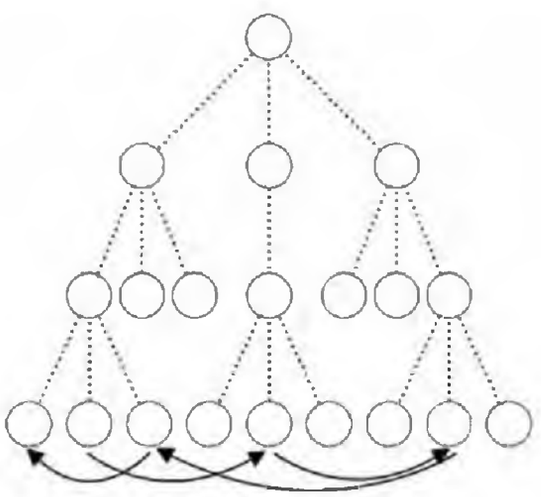

Рис. 4. Загальна схема роботи локального методу розв'язку CSP [4, с. 791]

[4] У наукових статтях англійською мовою термін «локальний» описується такими словами та виразами: local, reparative, iterative improvement.

```
reparative-search (a)
{
  while a is not solution
    a ← repair (a)
  return a
}
```

Алг. 2. Загальний алгоритм роботи алгоритму локального пошуку

Одним із недоліків локальних алгоритмів є їх неповнота, тобто у них пошук може зупинитися у локальному мінімумі, який насправді не є глобальним розв'язком. Для усунення цієї вади часто використовується рандомізація (із певною ймовірністю на кожному кроці обирається довільне рішення). Такі методи називаються стохастичними. Використовують й інші підходи: Randomized Iterative Improvement, еволюційні алгоритми, алгоритми на базі методу відпалювання, Tabu Search, динамічний пошук або алгоритми на основі штрафів, алгоритми з використанням принципу мурашиної колонії [2; 3; 7; 8; 12].

У загальному вигляді алгоритм локального пошуку для CSP-задач можна визначити таким чином (алг. 2): α – повний розв'язок; repair(α) – функція, що обирає наступний повний розв'язок. Схему роботи алгоритму відображено на рис. 1.

У таблиці 3 наведено основні алгоритми локального пошуку для розв'язку CSP задач. Вони розташовані (згори донизу) у порядку зростання ефективності. До того ж, майже кожен наступний із них ґрунтується на попередніх, що зазначено у колонці приміток кожного алгоритму. Інколи наведені алгоритми використовуються як «ідеї» для покращення конструктивних методів, що дає, як правило, суттєве покращення ефективності останніх. Втім про це докладніше буде сказано в наступному розділі.

У таблиці 4 наведено основні алгоритми для розв'язання розподілених CSP. Як бачимо, вони зовсім не повторюють класифікацію нерозподілених алгоритмів локального пошуку, окрім хіба що алгоритму DisBO, який, по суті, започаткував всю цю розподілену гілку.

Алгоритми локального пошуку прийнято порівнювати не тільки між собою, а й з алгоритмами конструктивного пошуку, адже ці гілки розвиваються паралельно і незалежно, і в кожній з них є представники, гідні бути суперниками. Головною перевагою конструктивних методів, звісно, лишається їх повнота, проте в деяких випадках вони виграють і в швидкодії. Тож перший розподілений алгоритм локального пошуку, DisBO порівнюється із конструктивним алго-

Таблиця 3. **Основні алгоритми локального пошуку для розв'язку CSP задач**

Клас алгоритму	Назва алгоритму	Короткий опис	Ефективність	Рік
Iterative Improvement	Min-conflict Heuristic (MCH)	На кожному кроці обирається випадкова змінна із конфліктного набору, і їй присвоюється нове значення, що мінімізує кількість незадоволених обмежень.	Добре працює для задачі про n ферзів з n близько мільйона.	1990
Randomised Iterative Improvement	The Min Conflicts Heuristic with Random Walk (WMCH)	На кожному кроці із певною ймовірністю вибирається змінна не із конфліктного набору.	Кращий від MCH.	1995
Tabu search	Min Conflicts Heuristic with Tabu Search (TMCH)	Використані значення стають забороненими для змінних на наступні n кроків.	Ефективніший від WMCH.	1997
	The Tabu Search Algorithm by Galinier and Hao (TS-GH)	Подібний до попереднього, але обрана змінна має мінімізувати кількість порушених обмежень.	Набагато ефективніший від TMCH.	1997
Penalty-based local search	GENET	Базується на нейронній мережі, елементами якої є атомарні значення змінних.	Порівняний із BO.	1991
	Breakout Method [12]	Використовує iterative improvement, доки не буде досягнуто локального мінімуму, а тоді накладає штраф 1 на всі незадоволені обмеження.	Приблизно настільки ж ефективний, як GENET.	1993

Таблиця 4. **Основні алгоритми локального пошуку для розв'язання задач DisCSP**

Назва алгоритму	Короткий опис	Ефективність	Рік
Distributed Breakout algorithm (DisBO) [7]	Накладає штраф на порушені обмеження у разі потрапляння у локальний мінімум.	Дає 15-кратне прискорення порівняно із ABT на складних задачах.	1996
DisBO-wd [1]	Здійснює переобчислення штрафів на кожному кроці за спеціальними формулами із коефіцієнтами, що підбираються до кожної задачі окремо.	Ненабагато швидший від DisBO.	2007
Distributed Guided Local Search (Dist-GLS) [2]	Використовує два види штрафів: temporary та incremental. Другий штраф використовується, як правило, після першого.	До 5 разів ефективніший від DisBO на будь-яких DCSP [2, с. 90].	2005
Distributed Penalty-Based Local search (DisPeL) [3]	Подібний до попереднього, але для обчислення значення, що мінімізує кількості незадоволених обмежень, задіяна спеціально визначена цільова функція.	До 6 разів ефективніший від DisBO [3, с. 50].	2005

ритмом AWCS. Автори статті DisBO порівнюють алгоритми на задачі про розфарбування графа. Для першого експерименту було згенеровано по 10 екземплярів DCSP для кожного n = {90, 120, 150}, кількість обмежень між агентами дорівнювала n x 2, а кількість доступних кольорів – три. На таблиці *5a* бачимо, що на всіх типах задач середня продуктивність алгоритму локального пошуку DisBO приблизно удвічі гірша від продуктивності конструктивного алгоритму.

Втім це не збентежило авторів алгоритму DisBO, і вони вирішили провести другий експеримент, використавши для тестів набагато складнішу задачу. Цього разу кількість обмежень між агентами становила n x 4,7, а кількість кольорів збільшилась до чотирьох. На таблиці *5b* бачимо, що для складних задач DisBO виявився значно ефективнішим за AWCS, причому ця залежність продуктивностей навіть не лінійна, і видно, що зі зростанням кількості вершин графа, здатність AWCS знайти розв'язок за розумний час зменшується. Втішає те, що ефективність мірялася в кількості кроків, потрібних алгоритму, автори ж DisBO чесно зізнаються, що їх алгоритм виконує значно більше обчислень на кожному кроці, ніж AWCS. Якщо в AWCS агент при виникненні конфлікту бере першого-ліпшого конфліктного сусіда і вирішує проблему з ним, то DisBO проводить ретельний аналіз перш ніж прийняти рішення, хто з його конфліктних сусідів є найбільш конфліктним, і в найгіршому випадку він мусить перебрати їх всіх. Але, навіть враховуючи це, автори DisBO все одно оцінюють свій алгоритм як більш продуктивний.

Таблиця 5a. **Порівняння продуктивності алгоритмів DisBO та AWCS на задачі про розфарбування графа з кількістю вершин n, кількістю кольорів 3 та кількістю обмежень n x 2**

n	DisBO		AWCS	
	% розв'язаних задач	к-ть кроків	% розв'язаних задач	к-ть кроків
90	100	150,8	100	70,1
120	100	210,1	100	106,4
150	100	278,8	100	159,2

Таблиця 5b. **Порівняння продуктивності алгоритмів DisBO та AWCS на задачі про розфарбування графа з кількістю вершин n, кількістю кольорів 4 та кількістю обмежень n x 4,7**

n	DisBO		AWCS	
	% розв'язаних задач	к-ть кроків	% розв'язаних задач	к-ть кроків
60	100	591,3	100	1733,6
90	100	1175,8	83	14897,3
120	100	2218,1	25	34771,6

Наступний алгоритм з таблиці, DisBO-wd, створено на основі DisBO, і метою його створення було не так загальне покращення алгоритму, як покращення його в напрямку розв'язання coarse-grained задач, тобто задач, де кожен агент володіє не однією, а декількома змінними.

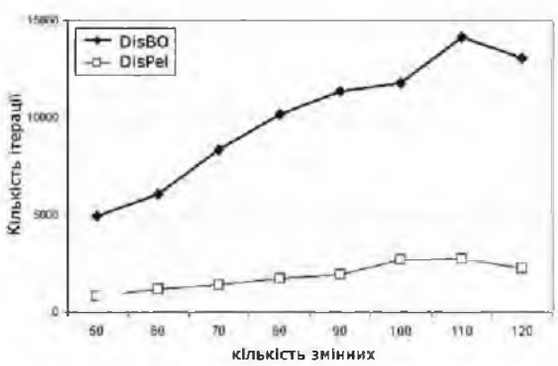

Рис. 5a. Кількість ітерацій, що знадобилася алгоритмам DisBO та DisPel для розв'язання DCSP з варійованою кількістю змінних

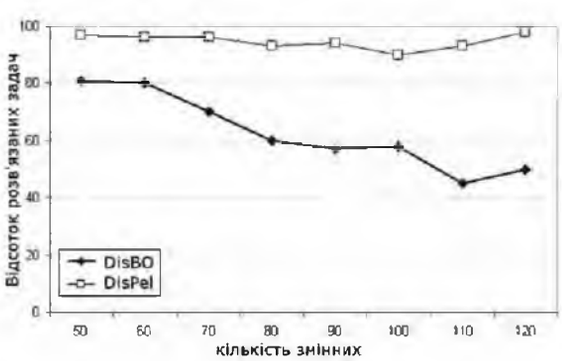

Рис. 5b. Відсоток задач, розв'язаних алгоритмами DisBO та DisPel, залежно від кількості змінних в задачі

Останні два алгоритми з таблиці зробили суттєвий прорив у швидкодії алгоритмів локального пошуку, а також можливим розв'язок дійсно дуже об'ємних і складних задач. До того ж, не зважаючи на те, що вони є неповними, відсоток їх невдач є відносно невеликим ~3–4 % за словами їх авторів, тож ці алгоритми набули широкого вжитку. Обидва ці алгоритми дуже схожі. Вони схожі настільки, що важко навіть знайти між ними відмінності. Вони вийшли в один рік і були розроблені одними і тими ж авторами. На рис. 5 (*a, b*) можна бачити

порівняння ефективностей DisPel та Dist-GLS та прослідкувати їхніми графіками кількість ітерацій, яка знадобилася кожному з них для розв'язку довільно згенерованих DCSP задач з певною кількістю змінних, а також відсоток розв'язаних ними задач.

1.3. Гібридні алгоритми

Гібридні алгоритми використовують комбінацію декількох різних підходів до розв'язання задачі. В контексті DisCSP гібриди можуть поєднувати у собі алгоритми з різних класів (локальний і конструктивний) та різного способу синхронізації (синхронні та асинхронні). Загальноприйнятої класифікації цих алгоритмів немає. У роботах [4; 5; 6; 9; 10; 11; 15] зустрічається класифікація гібридних алгоритмів в міру інтеграції локальної та конструктивної складових:

- низький рівень (локальний пошук виконується перед або після конструктивного пошуку);
- середній рівень (основою є локальний пошук, який користується засобами конструктивного алгоритму для зменшення простору пошуку);
- високий рівень (оператори конструктивного та локального пошуку застосовуються по черзі).

Серед алгоритмів із наведеної таблиці 6 до першого класу можна з упевненістю віднести Multi-Hyb, який поєднує у собі три алгоритми, що виконуються абсолютно незалежно один від одного, кожен самостійно намагається знайти рішення проблеми. До другого класу можна віднести алгоритми PenDHyb та DisBOBT, в яких результати локального пошуку використовуються як орієнтири для конструктивного алгоритму. ABT-Hyb є прикладом гібриду, що об'єднує два конструктивних алгоритми різного способу синхронізації, тож він не входить до наведеної класифікації, але за рівнем інтеграції компонентів його можна було б віднести до третього класу. Загальну схему алгоритмів третього класу описано у [13, с. 29].

Метою гібридів є компенсація недоліків однієї з компонент за рахунок іншої. Прикладом може бути поєднання синхронного методу, що характеризується послідовним виконанням (що є мінусом) і відносно низьким рівнем спілкування між агентами (це є плюсом), та асинхронного методу із протилежними характеристиками. Такий гібрид витрачає мінімум ресурсів на спілкування між агентами та користується перевагами паралельного виконання.

Таблиця 6. **Гібридні алгоритми розв'язку DisCSP**

Назва алгоритму	Компоненти алгоритму	Рік
ABT-Hyb [4]	Asynchronous Backtracking + Synchronous Conflict-Based Backjumping	2004
DisBOBT [6]	Distributed Breakout algorithm + Synchronous Backtracking	2003
PenDHyb [10]	Penalty-Based Local search + Synchronous Conflict-Based Backjumping	2008
Multi-Hyb [11]	Synchronous Exhaustive Backjumping + Penalty-Based Local search + Synchronous Conflict-Based Backjumping	2009
GT+Dis [15]	Synchronous Backtracking + Min-Conflict Heuristic (or one of the other methods from the table 2)	2003
GLoSS [21]	Guided Local & Systematic Search	2011

Висновки

Багато задач, пов'язаних із задоволенням обмежень та оптимізації, виникають в ситуаціях, коли кількість агентів різко збільшується і навіть може стати необмеженою. Тоді неможливо розв'язати проблему за рахунок централізації інформації на одному сервері. Розподілена задача задоволення обмежень допоможе при розв'язанні саме таких природно необмежених проблем. Локалізований характер задоволення обмежень є важливою перевагою в таких умовах, коли класичні методи оптимізації, як, наприклад, лінійне програмування, важко через високий динамізм застосовувати в розподіленому і, можливо, асинхронному режимі. Застосування розподілених алгоритмів задоволення обмежень тільки починають з'являтися. Тому різні алгоритми розв'язання описаних задач треба досліджувати, бо аналіз і класифікація таких алгоритмів стають джерелом пошуку оптимальних рішень.

Список літератури

1. Basharu M. DisBO-wd: a distributed constraint satisfaction algorithm for coarse-grained distributed problems / M. Basharu, I. Arana, H. Ahriz – Cambridge : Research and Development in Intelligent Systems XXIV // Proceedings of the 27th SGAI International Conference on Artificial Intelligence, AI-07, 2007. – P. 23–36.
2. Basharu M. Distributed Guided Local Search for Solving Binary DisCSPs / M. Basharu, I. Arana, H. Ahriz // Proceedings of the 20th national conference on Artificial intelligence, 2005. – P. 660–665.
3. Basharu M. Solving DisCSPs with Penalty Driven Search / M. Basharu, I. Arana, H. Ahriz // Proceedings of the 20th national conference on Artificial intelligence, 2005 – Vol. 1. – P. 47–52.
4. Brito I. Synchronous, Asynchronous and Hybrid Algorithms for DisCSP / I. Brito // Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP-2004). Lecture Notes in Computer Science, 2004 – Vol. 3258. – P. 791.
5. Chatzikokolakis K. Construction and Repair: A Hybrid Approach to Search in CSPs / K. Chatzikokolakis, G. Boukeas, P. Stamatopoulos / Eds. : G.A. Vouros and T. Panayiotopoulos : SETN 2004, LNAI 3025, 2004. – P. 342–351.
6. Eisenberg C. Hybrid Solving Scheme for Distributed Constraint Satisfaction Problems / C. Eisenberg, B.Faltings // The Fourth International Workshop on Distributed Constraint Reasoning at IJCAI. – 2003. – P. 1374–1375.
7. Hirayama K. Distributed Breakout Algorithm for Solving Distributed Constraint Satisafaction Problems / K. Hirayama, M. Yokoo // Proceedings of the Second International Conference on Multi-Agent Systems, MIT Press, 1996. – P. 401–408.
8. Hoos H. Local search methods / H. Hoos, E. Tsang // Foundations of Artificial Intelligence, 2006. – Vol. 2. – P. 135–167.
9. Jussien N. Local search with constraint propagation and conflict-based heuristics / N. Jussien, O. Lhomme // Artificial Intelligence. – 2002. – Vol. 139. – P. 21–45.
10. Lee D. A Hybrid Approach to Distributed Constraint Satisfaction / D. Lee, I. Arana, H. Ahriz, K. Hui // Proceedings of the 13th international conference on Artificial Intelligence. – 2008. – P. 375–379.
11. Lee D. Multi-hyb: a hybrid algorithm for solving DisCSPs with complex local problems / D. Lee, I. Arana, H. Ahriz, K.-Y. Hui // Web Intelligence and Intelligent Agent Technologies ; eds. P. Boldi, G. Vizzari, G. Pasi and R. Baeza-Yates. – 2009. – P. 379–382.
12. Morris P. The Breakout method for escaping from the local minima / P. Morris // Proceedings of the Eleventh National Conference on Artificial Intelligence. – 1993. – P. 40–45.
13. Nareyek A. Integrating local-search advice into refine-ment search (or not) / A. Nareyek, S. F. Smith, C. M. Ohler // Proceedings of the CP 2003 Third International Workshop on Cooperative Solvers in Constraint Programming. – 2003. – P. 29–43.
14. Prosser P. Binary constraint satisfaction problems: some are harder than others / P. Prosser // Proc. ECAI-94. – 1994. – P. 95–99.
15. Salido M. F. Stochastic Local Search for Distributed Constraint Satisfaction Problems / M. F. Salido, F. Barber // Proceeding of IJCAI Workshop on Stochastic Search Algorithms, Acapulco. – México, 2003. – P. 49–54.
16. Silaghi M.-C. Asynchronous aggregation and consistency in distributed constraint satisfaction / M.-C. Silaghi, B. Faltings // Artificial Intelligence, 2005. – Vol. 161, Issues 1–2. – P. 25–53.
17. Yokoo M. Asynchronous Weak-commitment Search for solving Distributed Constraint Satisfaction Problems / M.Yokoo // Proc. of the First International Conference on Principles and Practice of Constraint Programming. – 1995. – P. 88–102.
18. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving / M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara // Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems. – 1992. – P. 614–621.
19. Yokoo M. Distributed Constraint Satisfaction Algorithm for Complex Local Problems / M. Yokoo, K. Hirayama // Third International Conference on Multiagent Systems (ICMAS-98). – 1998. – P. 372–379.
20. Zivan R. Synchronous and Asynchronous Search on DisCSPs / R. Zivan, A. Meisels // Proc. of EUMAS-2003. – Oxford, UK, 2003. – P. 61–88.
21. Галковская Л. Гибридный алгоритм решения задачи удовлетворения ограничений / Л. Галковская, М. Глибовец, С. Гороховський // Информационные технологии. – Управляющие системы и машины. – ноябрь-декабрь. – 2012. – № 6. – С. 72–87.

L. Galkovska

ALGORITHMS FOR SOLVING DISTRIBUTED CONSTRAINT SATISFACTION PROBLEM

This work introduces the new method of classification of the algorithms for solving Distributed Constraint Satisfaction Problem. Classification is supported by examples, organized in comparison tables that, besides comparison of productivity of algorithms, also contain some recommendations on application of those solving methods to specific instances of DCSP.

Keywords: Constraint Satisfaction Problem, Distributed Constraint Satisfaction Problem, local search, reparative search, iterative improvement search, exhaustive search, global search, systematic search, constructive search, refinement search, hybrid algorithms.

Матеріал надійшов 10.09.2013