FRI-2G.303-1-CCT1-06

DECLARATIVE IMPLEMENTATIONS OF OPTIMIZATION METHODS FOR SOLVING TRAVELING SALESMAN PROBLEM[6]

Assist. Prof. Emilia Golemanova, PhD
Department of Computer Systems and Technologies,
"Angel Kanchev" University of Ruse
Tel.: 082-888-681
E-mail: EGolemanova@uni-ruse.bg

Assist. Prof. Tzanko Golemanov, PhD
Department of Computer Systems and Technologies,
"Angel Kanchev" University of Ruse
Tel.: 082-888-681
E-mail: TGolemanov@uni-ruse.bg

Abstract: *The paper presents three declarative solutions to a well-known optimization problem - Traveling Salesman Problem (TSP). It provides implementations of representative algorithms of the two main approaches to deal practically with NP-hard computational problems – exact and approximate optimization methods. It is demonstrated how the algorithms Branch-and-Bound, Nearest-Neighbor, and Simulated Annealing can be implemented in a declarative (i.e. an automatic) way, using the style of programming, named Control Network Programming (CNP). Because these implementations correspond to the graphical representation of TSP, they are very intuitive and easily programmable. Respectively, CNP and its supporting programming language Spider, can be used for teaching these exact, greedy, and metaheuristic algorithms.*

Keywords: *Declarative Programming, Traveling Salesman Problem, Branch-and-Bound, Nearest-Neighbor, Simulated Annealing*

ВЪВЕДЕНИЕ

Задачата за Пътуващия търговец (*The Traveling Salesperson, TSP*) е обект както на научни изследвания, така и на практически приложения. Една от причините за това е, че тя е концептуално проста, но същевременно и NP-трудна задача. Това я прави добър пример в теоретичната алгоритмика и повечето комбинаторни оптимизационни алгоритми са разработени с помощта на TSP. Трябва да се отбележи, че TSP е не само идеален пример за изучаване и тестване на различни оптимизационни методи. Съществуват редица реални задачи, които могат да се моделират и решат като нея.

В настоящия доклад се дискутират имплементации на три представителни алгоритъма на двата основни подхода за практическо справяне с NP-трудни изчислителни задачи – точни и приближени оптимизационни методи. Демонстрира се как алгоритмите *Branch-and-Bound*, *Nearest-Neighbor* и *Simulated Annealing* могат да бъдат реализирани по декларативен (т.е. автоматичен) начин, използвайки визуално-графичния стил на програмиране *Control Network Programming (CNP)*. Тъй като тези реализации съответстват на графичното представяне на TSP, те са много интуитивни и лесно програмируеми. Съответно CNP и поддържащият го език за програмиране Spider могат да се използват за преподаване на тези точни, алчни и метаевристични алгоритми.

ИЗЛОЖЕНИЕ

Съществуват два основни подхода за справяне с NP-трудни комбинаторни задачи (Talbi, 2009):

[6] Докладът е представен на научната сесия на 27.10.2023 в секция „Комуникационна и компютърна техника" с оригинално заглавие: DECLARATIVE IMPLEMENTATIONS OF OPTIMIZATION METHODS FOR SOLVING TRAVELING SALESMAN PROBLEM

- Използва се стратегия, която гарантира решаването на задачата точно, но не гарантира намирането на решение за полиномно време. Тези методи в Теория на оптимизацията се наричат **точни** (*exact, precise*), а в Изкуствения интелект - пълни (*complete*);
- Използва се алгоритъм, който може да намери приближено (за оптимизационните задачи - субоптимално) решение за полиномно време. Тези методи се наричат **приближени** (*approximate*) оптимизационни методи, често наричани още **евристични** (*heuristic*).

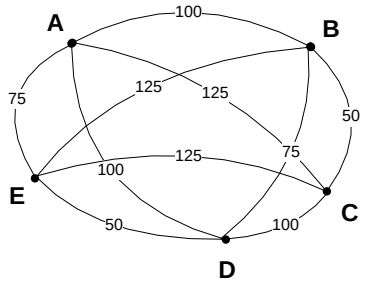
И точните, и приближени оптимизационни методи могат да бъдат разглеждани като стратегии за търсене в граф – графа на пространство на състоянията на задачата или графа на съседство на задачата. Бидейки алгоритми за обхождане на алтернативи, евристичен избор на някои от тях и „отсичане“ на други, имплементирането на тези оптимизационни алгоритми не е тривиална задача, особено ако те са предмет на разработка по време на учебен процес.

Програмирането с управляващи мрежи *(Control Network Programming, CNP)*

Програмирането с управляващи мрежи (Kratchanov, Golemanov and Golemanova, 2007), или накратко CNP, е авторска разработка, представляващо стил за декларативно визуално-графично програмиране. Програмата в CNP е множество от рекурсивни графи, наречена управляваща мрежа (CN), която се разработва и визуализира в графичен вид. „Изчислението” в CNP е всъщност „търсене“, т.е. изпълнението на CNP-програма е търсене на път в това множество от рекурсивни графи, което, в допълнение, може да бъде управлявано от вградени системни средства (Kratchanov, T.Golemanov and E.Golemanova, 2009), (Kratchanov *et al.*, 2010). Задачите, които могат естествено да се визуализират като графоподобни структури, са основни кандидати за ефективни и прости CNP-решения. Конкретен пример е случаят, когато представянето на задачата се основава на подходите на Пространството на състоянията (ПС) или Графа на съседство (ГС). Обикновено в Spider - езикът за CNP (Golemanov, Kratchanov and Golemanova, 2000), при решаването на такава задача, управляващата мрежа, т.е. програмата „копира“ това графоподобно представяне, което е недетерминирано, и не се изисква разработването на експлицитна процедура, реализираща процеса на търсене на решение и разрешаваща недетерминизма. Вместо това, вграденият механизъм на търсене, заедно с богатата палитра от управляващи го системни средства, извършват избраната стратегия, която може да бъде дори евристична или стохастична, търсейки път между началния и финален възел на CN. Така, Spider дава възможност на програмиста за „автоматично“, т.е. декларативно решаване на задачата. Получените програми са лесни за разработване и разбиране, което е важно за Изкуствения интелект и Теория на оптимизацията, където алгоритмите обикновено са недетерминирани, евристични или стохастични. Съответно CNP и поддържащият го език за програмиране Spider могат да се използват за преподаване на тези алгоритми (Kratchanov *et al.*, 2012).

Пътуващ търговец (*The Traveling Salesperson, TSP*)

Ще бъде разгледан следният екземпляр на *TSP* (Фиг. 1), дефиниран в (Luger, 2009):



Фиг.1. Задача за Пътуващия търговец

Графичното представяне на тази задача (Фиг. 1) съвпада с нейното ПС, т.е. самата карта е ПС на задачата. То е от типа „просто ПС“. При него състоянието няма вътрешна структура, а е атомарен обект (token). Характерно за този вид ПС е, че то представя физическата среда на задачата, т.е. ПС всъщност е „снимка на света”. Подобно ПС имат т.нар. Задачи за маршрутизиране (routing problems).

Най-интуитивният и лесен подход за решаване на тази задача е картата да се „трансформира“ в CN и вграденият механизъм на търсене да намери оптимален ацикличен път между начално и финално състояние, съответстващи на зададения начален град.

Алгоритъм Branch-and-Bound

Най-често *Branch-and-bound* генерира възлите дървото на ПС според така нареченото *best-first*-правило (Levitin, 2012). Тук ще бъде разгледана неговата версия, при която ПС се обхожда в дълбочина, тъй като ще се използва вграденият механизъм за търсене на *Spider,* който е вариант на *Bakctracking*-алгоритъма.

Предложеният подход от Фиг. 2 може да се разглежда като вариант на решението на TSP чрез алгоритъма *Branch-and-Bound*, описано в (Levitin, 2012). Разликата е в използваната функция за пресмятане на долната граница на целевата функция, т.е. на долната граница на дължината на тура. *Spider*-решението от Фиг. 2Фиг изчислява долната граница на тур, минаващ през даден град, като дължина на изминатия до него път. Всички пътища, продължаващи от този град нататък по картата ще бъдат с по-голяма дължина. Следователно изчислената по този начин стойност наистина е долна граница на всички турове, включващи този изминат път.

Така, Фиг. 2 реализира идеята, да се разглеждат се само тези ациклични турове, чиито разходи не надхвърлят една динамично изменяща се граница. Цената на всяко новооткрито решение става стойност на системната опция MAXPATHCOST. Така, последното открито решение е оптималното.

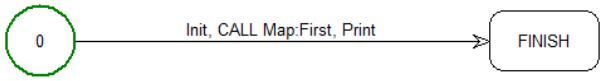
В примитива *Init* на главната мрежа се въвежда началния град в глобалната променлива *First* и се инициализира променливата *Max,* която съхранява дължината на най-доброто до момента решение. Следва преход към подмрежа *Map* в начално състояние *First.*

Подмрежата *Map* съответства на пълния *г*раф, описващ градовете и разстоянията между тях, т.е. на простото ПС на задачата. Разстоянията са зададени като разходи по стрелките на подмрежата. По този начин програмата автоматично изчислява дължината на текущия път, т.е. долната граница на целевата функция за всяко състояние от ПС. Ако тази информация е налична, алгоритъмът *Branch-and-Bound* изисква сравняване на тази граница със стойността на най-доброто до момента решение и ако границата не е „по-добра“, т.е. не е по-малка, състоянието е неперспективно и този клон на ПС може да бъде „отрязан“. Това означава, че развитието на това състояние няма да доведе до по-добро решение от вече намереното такова. Следователно стойността най-доброто до момента решение играе ролята на горна граница на всички следващи решения. В *Spider* тази идея лесно се моделира чрез системната опция MAXPATHCOST. Обновяването на стойността й със стойността на следващото решение се реализира благодарение на възможността стойност на тази опция да е променлива. В конкретната реализация това е променливата *Max*, която се обновява в примитива *Print*.

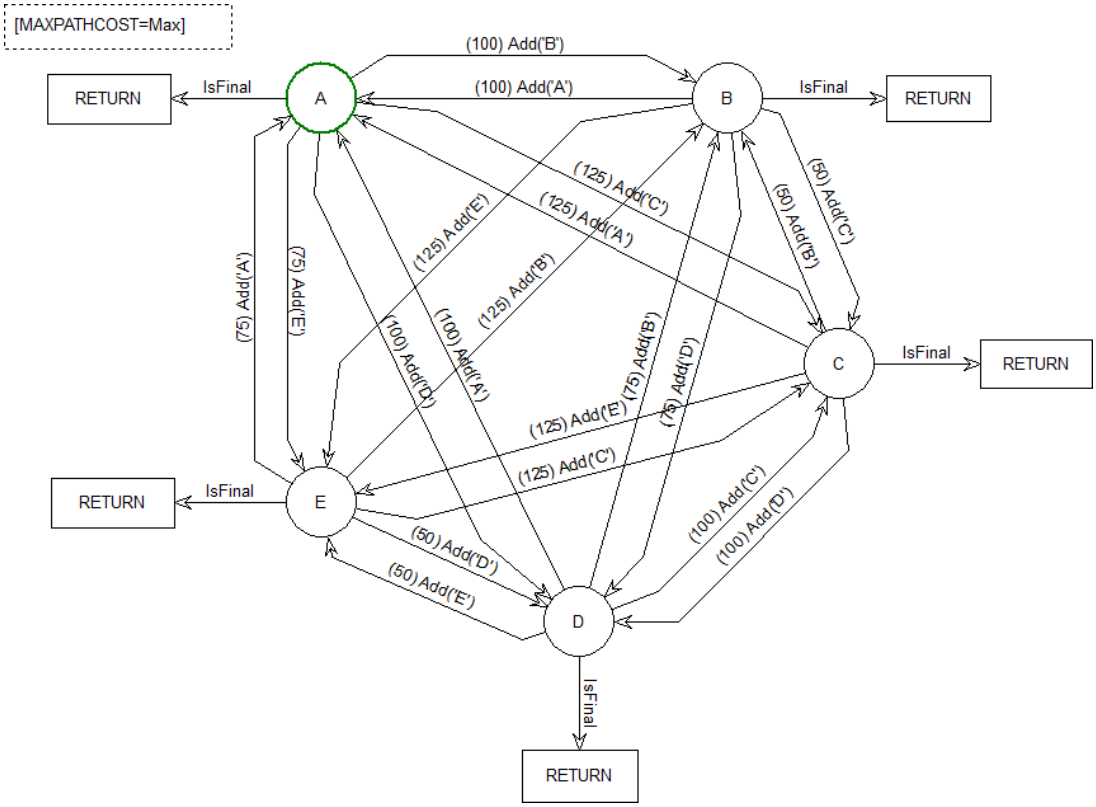
Основната задача на примитива *Add(City)* е както в решението от 1.1.6 – добавя *City* (при обратно изпълнение – премахва) от потребителски дефинирания стек на пътя на решение. В допълнение, този примитив извършва и проверката за ацикличност на пътя, която осигурява ограничението на задачата за търсене на тур, минаващ през всеки град по веднъж (с изключение на първия град).

[ARROWCOST=0, SOLUTIONS=ALL]

Main TSP



Sub Map



Фиг. 2. *Spider*-решение на TSP - *Branch-and-Bound*

Алгоритъм Nearest-Neighbor

Алгоритъмът *Nearest-Neighbor* е един от най-простите апроксимиращи алгоритми за решаване на TSP, който е базиран на алчната техника за дизайн на алгоритми. Този добре известен алчен алгоритъм използва евристиката „*най-близък съсед*“, чийто принцип е: „винаги върви до най-близкия непосетен град“.

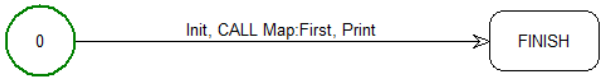
Декларативното *Spider*-решение на този алгоритъм е представено на Фиг. 3.

Главната подмрежа *TSP* извършва инициализация, извикване на подмрежата за търсене на решение и извежда намереното решение и неговата дължина.

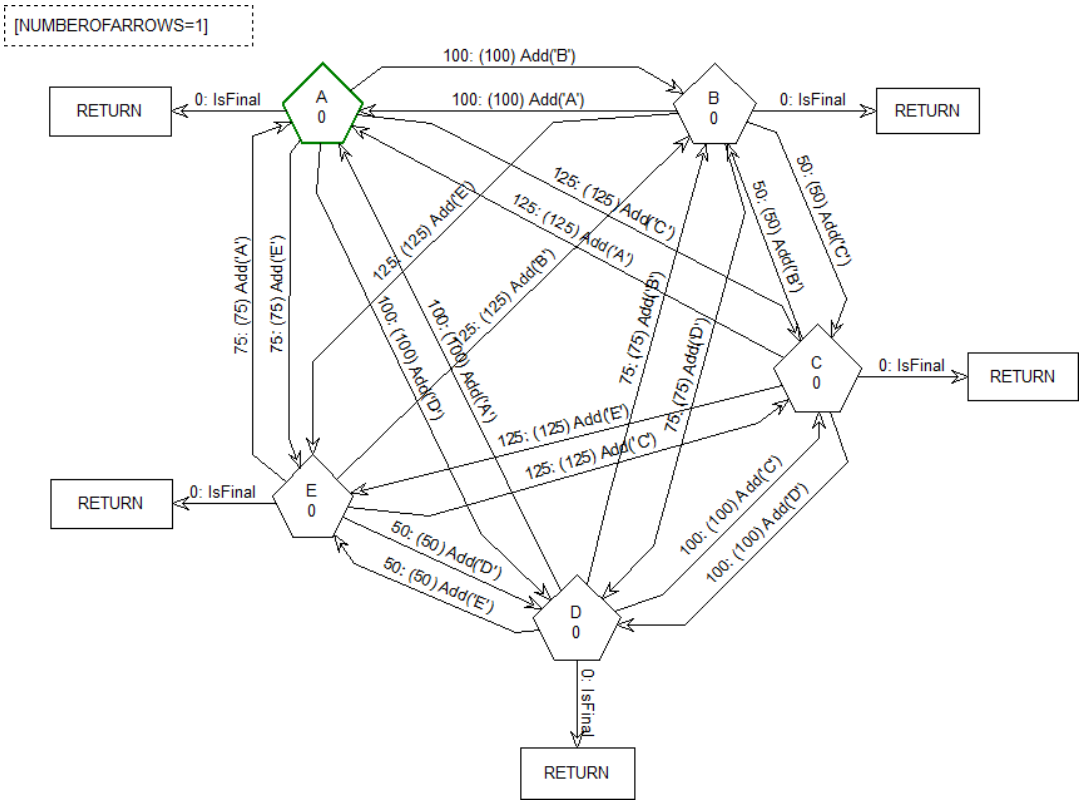
Подмрежата *Map* отново е декларативното представяне на задачата и съответства на нейното графично описание от Фиг. 1. Тъй като използваната евристика изисква намиране на най-близкия град, всеки град трябва да се представи като управляващо състояние, чиито излизащи стрелки да бъдат преподредени според разстоянията до съседните му градове. Затова тези числови стойности се използват като оценки на стрелките. Използва се управляващо състояние от тип ORDER със селектор 0. Така излизащите от него стрелки ще бъдат подредени „по близост спрямо 0“, т.е. във възходящ ред. Първа ще бъде изпробвана стрелката, проверяваща дали е намерен легитимен тур, тъй като нейната оценка е 0. При успешно изпълнение на примитива *IsFinal(City)* се излиза от подмрежата *Map* и се извежда намереният тур и неговата дължина. Изчислението на дължината на тура става, използвайки разстоянията между градовете като „разходи“ по стрелките на *Map*.

Ако примитивът *IsFinal(City)* е неуспешно изпълнен, управлението се връща назад и се изпробва стрелката, съответстваща на най-късото ребро, т.е. изпробва се преход към най-близкия съсед на *City*. Примитивът *Add(City)* има същите функции както в горното решение от Фиг. 2 – предотвратява цикъл с по-малка дължина от търсения тур и добавя текущия град към решението. Алгоритъмът *Nearest-Neighbor* е стратегия без възврат (*irrevocable*), което на *Spider* се постига чрез използване на системната опция [NUMBEROFARROWS=1]. Тя „отрязва“ останалите стрелки от състоянието при първа успешно изпълнена стрелка.

Main TSP



Sub Map



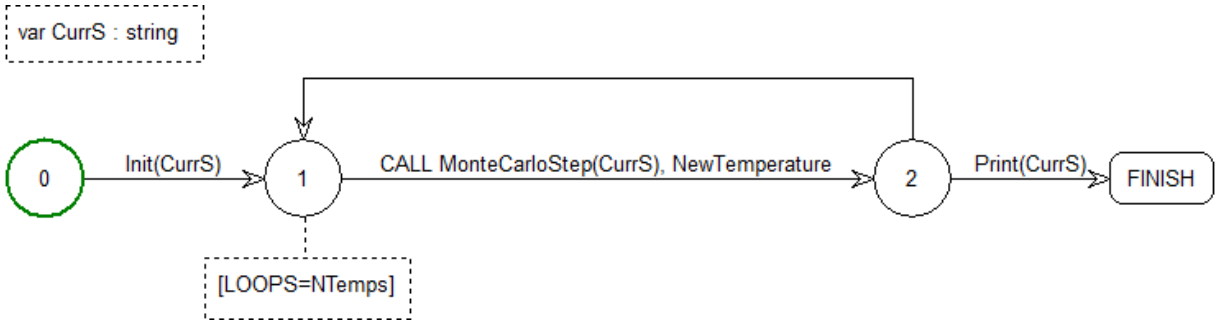
Фиг. 3: *Spider*-решение на TSP - *Nearest-Neighbor*

Алгоритъм Simulated Annealing

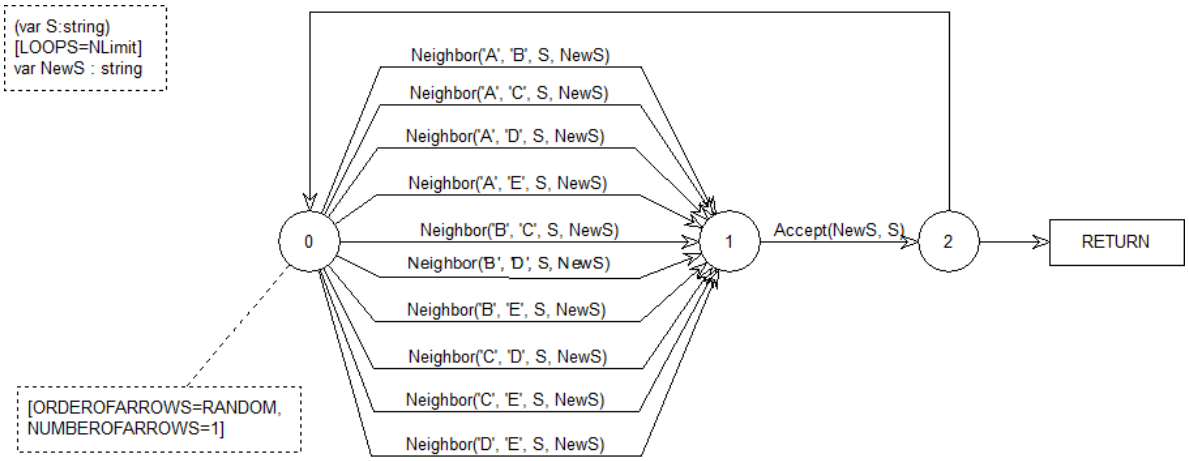
Spider-програмата Фиг. 4 имплементира метаевристичния алгоритъм *Симулирано закаляване (Simulated Annealing)*, дефиниран със следния псевдо-код:

```
initialize temperature
for i := 1...ntemps do
    temperature := factor * temperature
    for j := 1...nlimit do
        try swapping a random pair of points
        delta := current_cost - trial_cost
        if delta > 0 then
            make the swap permanent
        else
            p := random number in range [0...1]
            m := exp( delta / temperature )
            if p < m then                                  // Metropolis criterion
                make the swap permanent
            end if
       end if
    end for
end for
```

Main SimulatedAnnealing



Sub MonteCarloStep



Фиг. 4. *Spider*-решение на TSP - *Simulated Annealing*

Симулираното закаляване е метаевристика, базирана на *Metropolis*-евристиката. Подмрежата *MonteCarloStep* реализира Локално търсене с тази евристика чрез декларативно описание на едно ниво от ПС. Състояние 0, съответства на абстрактно състояние от ПС, а излизащите от него стрелки моделират функцията за генериране на наследници. Външният цикъл се имплементира чрез главната мрежа, използвайки системната опция LOOPS. Стойностите на *NTemps* (брой температурни промени) и *NLimit* (брой опити при дадена температура) в *Spider*-решението трябва да бъдат с 1 по-малки от съответните стойности в псевдо-кода, тъй като опцията LOOPS определя максималния брой повторни влизания в състояние на CN. Примитивът *Neighbor* и системната опция [ORDEROFARROWS=RANDOM] реализират действието *try swapping a random pair of points*, а примитивът *Accept* е имплементация на *Metropolis*-евристиката.

ЗАКЛЮЧЕНИЕ

Представеният декларативен подход за решаване на задачи чрез търсене в ПС се характеризира с това, че CN описва естественото графично представяне на задачата. Така CN е или простото ПС на задачата, или ако нейното ПС не е просто, то CN го генерира и търси в него. Не се изисква разработването (в общоприетия му смисъл) на алгоритъм за търсене в ПС – поведението му се постига „автоматично", благодарение на вградения механизъм на търсене в *Spider* и средствата за неговото управление. В резултат, тези „автоматични" реализации на алгоритми за търсене са интуитивни и естествени и съответстват на човешкия начин на мислене и специфициране на сложни задачи. Средствата за статично и динамично управление на механизма на търсене в *Spider* се оказват удобен и мощен инструмент за реализация на богат набор от слепи, евристични и метаевристични алгоритми за търсене в

ПС. Освен това едно CNP-решение, реализиращо даден алгоритъм лесно може да бъде модифицирано до друг алгоритъм или до същия, но с други параметри, пренастройвайки управляващата му мрежа чрез използване на други системни средства. Това определя предлагания метод за разработване на декларативни имплементации чрез *Spider* като адаптивен и гъвкав.

ACKNOWLEDGEMENTS

Този доклад се публикува с подкрепата на 23-ФЕЕА-01 „Разработване на модели и симулации с различни области на приложение“, финансиран от фонд „Научни изследвания“ на Русенския университет „Ангел Кънчев“.

REFERENCES

Golemanov, T., Kratchanov, K. and Golemanova, E. (2000) SPIDER – A Language for Programming Through Control Networks', in *CompSysTech 2000*. Sofia, Bulgaria: ACM Press, pp. 2091–2095.

Kratchanov, K. *et al.* (2010) 'Control Network Programming with SPIDER: Dynamic Search Control', in *14th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2010)*. Cardiff, UK, pp. 253–262. doi: 10.1007/978-3-642-15390-7_26.

Kratchanov, K. *et al.* (2012) 'Using Control Network Programming in Teaching Randomization', in *International Conference on Electronics, Information and Communication Engineering (EICE 2012), Macau, China*. Macau, China, pp. 67–71.

Kratchanov, K., Golemanov, T. and Golemanova, E. (2007) 'Control Network Programming', in *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*. Melbourne, Australia, pp. 1012–1018. doi: 10.1109/ICIS.2007.85.

Kratchanov, K., T.Golemanov and E.Golemanova (2009) 'Control Network Programs: Static Search Control with System Options', in *8th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED 2009), Cambridge, UK*. WSEAS Press (2009), pp. 423–428.

Levitin, A. (2012) *Introduction to Design and Analysis of Algorithms*. Third. Pearson.

Luger, G. (2009) *Artificial Intelligence: Structures and Strategies for Complex Problem Solving (6th Edition)*. Pearson.

Talbi, E.-G. (2009) *METAHEURISTICS: FROM DESIGN TO IMPLEMENTATION*. John Wiley & Sons, Inc.