GMAS-1DMCSP: HỆ THỐNG ĐA TÁC TỬ GEN GIẢI BÀI TOÁN CẮT VẬT TƯ MỘT CHIỀU VỚI NHIỀU KÍCH THƯỚC VẬT LIỆU THÔ

PHAN THỊ HOÀI PHƯƠNG, LƯƠNG CHI MAI

1. GIỚI THIỆU

Bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô (*One-Dimensional Cutting Stock Problem with Multiple Stock Sizes)* là bài toán tối ưu thuộc lớp bài toán NP-Hard. Trong thời gian gần đây đã có nhiều công trình đề cập tới các giải pháp cho bài toán này, trong đó có [1, 10, 11, 12, 13, 16, 18]. Do bài toán có độ phức tạp tính toán lớn nên việc tăng tốc độ tính toán cho các giải pháp mang một ý nghĩa quan trọng khi áp dụng vào thực tế.

Bài báo này đề xuất xây dựng một hệ thống đa tác tử để nâng cao hiệu quả giải bài toán cắt vật tư với nhiều kích thước vật liệu thô trên cơ sở thuật toán GA-CG được công bố trong [1]. Với hệ thống này, một mặt các công việc tính toán của thuật toán GA-CG được song song hóa và phân chia cho các tác tử khác nhau đảm nhiệm. Điều đó cho phép giảm thiểu một cách đáng kể thời gian thực hiện thuật toán; Mặt khác các tác tử được phân bố trên toàn bộ các tài nguyên tính toán có thể có của mạng cục bộ nên tận dụng được sức mạnh tính toán của tài nguyên và vì vậy thích hợp cho việc giải các bài toán thực tế có kích thước rất lớn.

Hệ thống này được thiết kế theo kiến trúc A-Team trên nền tảng JADE (Java Agent DEvelopment Framework) – một phần mềm trung gian (middle-ware) nguồn mở hỗ trợ cho việc phát triển các hệ thống đa tác tử được cài đặt hoàn toàn bằng Java và tuân thủ chặt chẽ các đặc tả của FIPA (The Foundation of Intelligent Physical Agents). Phần còn lại của bài được cấu trúc như sau. Mục 2 dành cho việc phát biểu bài toán cắt vật tư một chiều mở rộng (nhiều loại vật liệu thô) và trình bầy tóm tắt các kết quả lí thuyết về mối liên quan ngữ nghĩa của bài toán mở rộng này với bài toán cắt vật tư một chiều kinh điển (một loại vật tư) làm cơ sở đề xuất thuật toán phân tán giải bài toán cắt vật tư mở rộng. Thiết kế và cài đặt của hệ thống đa tác tử giải bài toán cắt vật tư mở rộng trên nền tảng JADE dựa trên thuật toán đề xuất trong Mục 2 được trình bày trong Mục 3. Mục 4 đưa ra một số đánh giá hệ thống trong môi trường ứng dụng thực tế. Cuối cùng là một số kết luận của bài báo.

2. BÀI TOÁN CẮT VẬT TƯ MỘT CHIỀU VỚI NHIỀU KÍCH THƯỚC VẬT TƯ VÀ THUẬT TOÁN GIẢI

Bài toán cắt vật tư một chiều với một loại vật liệu thô (bài toán kinh điển) được xác định bởi các dữ liệu sau: $(m, L, l = (l_1,...,l_m), b = (b_1,...,b_m))$, trong đó L là bề rộng của tấm vật liệu thô, m là số dạng vật liệu thành phẩm được cắt từ vật liệu thô và đối với mỗi dạng vật liệu thành phẩm j, l_j là bề rộng và b_j là đơn hàng cho loại vật liệu thành phẩm đó. Bài toán đặt ra là tìm cách cắt sao cho số lượng tấm vật liệu thô sử dụng là ít nhất mà vẫn đáp ứng được đơn hàng.

Gilmore và Gomory [14, 15] lần đầu tiên đưa kĩ thuật tạo sinh cột (Column Generation) do Ford và Fulkerson đề xuất và sau đó được Dantzig và Wolfe hoàn thiện vào ứng dụng trong thực tế với việc giải bài toán cắt vật tư kinh điển này. Có thể tóm tắt phương pháp giải bài toán cắt vật tư một chiều của Gilmore và Gomory như sau.

Một phương án cắt được biểu diễn bởi một vectơ cột $a^j = (a_{1j},...,a_{mj}) \in Z_+^m, j = 1,...,n$ sẽ cho biết số lượng vật tư thành phẩm được cắt từ tấm vật liệu thô. Phương án cắt được chấp nhận nếu nó thỏa mãn điều kiện:

$$\sum_{i=1}^{m} l_i a_{ij} \le L . \tag{1}$$

Giả sử $x_j, j = 1,...,n$ là số lần phương án cắt chấp nhận được được sử dụng trong nghiệm. Mô hình của Gilmore và Gomory trở thành:

$$1DCSP_G(m,L,l,b) = \min \sum_{j=1}^{n} x_j = \min \|x\| \tag{2}$$

Trên miền ràng buộc:

$$\sum_{j=1}^{n} a_{ij} x_j \ge b_j \qquad i=1,...,m \tag{3}$$

$$x_j \in Z_+, \qquad j=1,...,n \tag{4}$$

Mô hình (1)-(4) là bài toán quy hoạch nguyên với số lượng biến *n* tăng theo hàm lũy thừa của *m.* Mô hình này có suy yếu liên tục mạnh với tính chất làm tròn nguyên cải biên (*Modified Integer Round-Up Property* – **MIRUP**).

Trên cơ sở đó, Gilmore và Gomory đề xuất cách tiếp cận giải bài toán (1)-(4) gồm 2 bước: 1/ giải bài toán suy yếu liên tục của (1)-(4); 2/ Làm tròn số nghiệm tối ưu của bài toán suy yếu liên tục để nhận được nghiệm của bài toán (1)-(4).

Để giải bài toán suy yếu liên tục của (1)-(4) với số lượng biến *n* rất lớn, Gilmore và Gomory đề xuất sử dụng kĩ thuật tạo sinh cột (Column Generation) trong đó mỗi biến chỉ được sinh ra khi nó thực sự cần thiết cho việc cải thiện nghiệm tìm được trước đó. Sau khi thêm vào các biến bổ xung (slacks) ta có thể đưa bài toán (1)-(4) về dạng chuẩn:

$$1DCSP_G(m,L,l,b) = \min\{\|x\| : Ax = b, x \in Z_+^n\} \tag{5}$$

Suy yếu liên tục của (5) nhận được bằng việc loại bỏ ràng buộc nguyên trên các biến và được gọi là bài toán chủ (Master Problem - MP) sẽ có dạng:

$$1DCSP_G^{LP}(m,L,l,b) = \min\{\|x\| : Ax = b, x \in R_+^n\} \tag{6}$$

Xuất phát, kĩ thuật tạo sinh cột sẽ làm việc với một tập con các cột của A được gọi là *variable pool* hoặc *Restricted Master Problem* (RMP). RMP có thể được khởi tạo dễ dàng, ví dụ bởi cơ sở kiến thiết ban đầu của phương pháp đơn hình. Giả sử A' là các cột trong A được lựa chọn. Khi đó RMP có dạng:

$$1DCSP_G^{LP}(m,L,l,b) = \min\{\|x\| : A'x = b, x \in R_+^n\}. \tag{7}$$

Nhận xét rằng giá suy giảm của một biến ứng với phương án cắt chấp nhận được a trong (7) được xác định bởi công thức $1-ua$, với $u \in R_+^m$ là các giá trị biến đối ngẫu tối ưu của (7). Do vậy nghiệm tối ưu của bài toán xác định giá (*Pricing Problem*):

$$\max\{ua : al \le L, a \in Z_+^m\} \tag{8}$$

sẽ lần lượt được thêm vào RMP nếu nếu giá trị tối ưu của (8) lớn hơn 1. Nếu (8) không có nghiệm tối ưu như vậy thì nghiệm tối ưu của bài toán RMP (7) chính là nghiệm tối ưu của bài toán MP (6).

Sau khi đã giải được bài toán $1DCSP_G^{LP}(m,L,l,b)$, bước tiếp theo trong cách tiếp cận của Gilmore và Gomory là việc thực hiện thủ tục làn tròn số để nhận được nghiệm nguyên cho bài toán $1DCSP_G(m,L,l,b)$. Nhiều tác giả đã đưa ra các heuristics khác nhau để giải quyết vấn đề này. Một số thủ tục có thể tra cứu trong [10, 17].

Việc mở rộng bài toán cắt vật tư một chiều kinh điển trong trường hợp có nhiều kích thước vật liệu thô đã được một số tác giả đề cập đến [1, 10, 11, 12, 13, 16, 18]. Các tác giả đều thống nhất rằng việc xây dựng giải thuật heuristic cho bài toán là cần thiết vì các cách tiếp cận chính xác là không phù hợp trong trường hợp này.

Sau đây chúng ta sẽ trình bày tóm tắt một đề xuất lai ghép giữa thuật toán gen với phương pháp của Gilmore và Gomory để giải lớp bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô đã được công bố trong [1].

Bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô (*One-Dimensional Cutting Stock Problem with Multiple Stock Sizes* – 1DMCSP) là mở rộng tự nhiên của bài toán 1DCSP trong đó các tấm vật liệu thô có thể có kích thước khác nhau. Bài toán 1DMCSP có thể được đặc trưng bằng các dữ liệu sau:

$$(m, M, l = (l_1,...,l_m), b = (b_1,...,b_m), L = (L_1,...,L_M), c = (c_1,...,c_M)) \tag{9}$$

trong đó *m, l* và *b* mang ý nghĩa như trong bài toán 1DCSP; M là số các loại vật liệu thô sẽ được sử dụng và tham số L trong 1DCSP được thay thế bởi vectơ $L = (L_1,...,L_M)$, với L_i là bề rộng và c_i là giá của vật liệu thô thứ *i*, *i = 1,...,M.* Trong bài này ta giả thiết rằng giá của vật liệu thô tỉ lệ thuận với bề rộng của vật liệu.

Mô hình của bài toán trên (*Machine Balance Model*) được Gilmore và Gomory xây dựng trong [15]. Mô hình có thể được phát biểu như sau:

Đặt $m' = m + M$. Một vectơ $a = (a_1,...,a_{m'}) \in Z_+^{m'}$ là biểu diễn của một phương án cắt nếu

$$\sum_{i=1}^{m} l_i a_i \le \sum_{i=1}^{M} L_i a_{i+m} \quad \text{và} \quad \sum_{i=1}^{M} a_{i+m} = 1 \tag{10}$$

Các thành phần a_i, *i=1,...,m,* xác định có bao nhiêu vật tư thành phẩm loại *i* được cắt. Thành phần a_{k+m} sẽ bằng 1 nếu loại vật liệu thô thứ *k* được sử dụng còn các thành phần a_{i+m} với $i \in \{1,...,M\} \setminus k$ bằng 0.

Giả sử a_i, i=1,…,*n,* là tất cả các phương án cắt và mỗi thành phần x_i của vectơ $x = (x_1,...,x_n)$ là số lần sử dụng phương án cắt a_i. Nhận xét rằng *n* có thể rất lớn. Ta xây dựng

một vectơ *n* chiều c' từ *c* theo cách sau. Nếu phương án cắt a_i được thực hiện trên tấm thép loại *k* thì $c_i^{'} = c_k$, *i=1,...,n* và *k=1,...,M*. Khi đó ta có mô hình:

$$1DMCSP(m,M,l,b,L,c) = \min c'x \tag{11}$$

Trên miền ràng buộc:

$$\sum_{i=1}^{n} a_{ij}x_i \geq b_j, \qquad j=1,...,m \tag{12}$$

$$\sum_{j=1}^{m} l_j a_{ij} \leq \sum_{j=1}^{M} L_j a_{i(j+m)} \tag{13}$$

$$\sum_{j=1}^{M} a_{i(j+m)} = 1 \tag{14}$$

$$x \in Z_+^n \tag{15}$$

$$a_i \in Z_+^{m'} \qquad i=1,...,n \tag{16}$$

Từ các phát biểu (1)-(4) và (11)-(16) ta có mối liên hệ giữa hai bài toán như sau:

Định lí 1. [1] $1DMCSP(m,M,l,b,L,c) \leq 1DCSP_G(m,L_k,l,b)$ với mọi $k \in \{1,...,M\}$. *Nói cách khác, bài toán cắt vật tư với nhiều kích thước vật liệu thô sẽ tốt hơn việc cắt chỉ trên một loại vật liệu.*

Định lí 1 khẳng định ý nghĩa của việc xây dựng các giải pháp cho bài toán cắt vật tư với nhiều kích thước vật liệu thô.

Giả sử *x** là nghiệm tối ưu của (11)-(16). Ta ký hiệu:

- $\Omega(k)$, *k=1,...,M,* là tập tất cả các phương án cắt trên vật liệu thô thứ *k* được sử dụng trong (11)-(16) tương ứng *x**;

- $x^*/\Omega(k)$ là thu hẹp của *x** trên $\Omega(k)$;

- b^k là vectơ vật tư thành phẩm nhận được từ (11)-(16) với việc sử dụng số lần cắt theo các phương án cắt trong $\Omega(k)$ được xác định bằng các thành phần tương ứng của $x^*/\Omega(k)$.

Định lí 2. [1] *Nếu x* là nghiệm tối ưu của bài toán* $1DMCSP(m,M,l,b,L,c)$ (11)-(16) *thì* $x^*/\Omega(k)$ *là nghiệm tối ưu của bài toán* $1DCSP_G(m,L_k,l,b^k)$ (1)-(4), *k = 1,...,M.*

Trên cơ sở của Định lí 2, chúng ta có thể mô hình hóa bài toán 1DMCSP dưới dạng phát biểu mới như sau.

$$1DMCSP(m,M,l,b,L,c) = \min \sum_{k=1}^{M} 1DCSP_G(m,L_k,l,b^k)c_k. \tag{17}$$

Trên miền ràng buộc:

$$\sum_{k=1}^{M} b^k \geq b \tag{18}$$

$$b^k \in Z_+^m. \tag{19}$$

Định lí 3. [1] *Nghiệm tối ưu của bài toán* (17) - (19) *sẽ xác định trên tập các vectơ* b^k *thỏa mãn* $\sum_{k=1}^{M} b^k = b$. *Tập các vectơ* b^k *như vậy được gọi là phân hoạch của b. Nói cách khác nghiệm tối ưu của* (17) - (19) *được xác định trên tập các phân hoạch của vectơ b.*

Các định lí 2, 3 và phát biểu mới của bài toán *1DMCSP* trong (17) - (19) gợi cho chúng ta ý tưởng phân rã bài toán *1DMCSP* thành các bài toán cơ sở là bài toán $1DCSP_G(m, L_k, l, b^k)$ để có thể áp dụng được phương pháp giải sử dụng kĩ thuật tạo sinh cột của Gilmore và Gomory. Các nghiệm tối ưu của từng bài toán cơ sở sẽ được kết hợp lại để tạo nên nghiệm chấp nhận được của bài toán 1DMCSP. Việc tìm nghiệm tối ưu cho bài toán 1DMCSP sẽ do thuật toán gen đảm nhiệm với không gian tìm kiếm là tập các phân hoạch của vectơ đơn hàng.

Sau đây chúng ta sẽ lần lượt hình thức hóa ý tưởng đó trên ngôn ngữ của thuật toán gen và kĩ thuật tạo sinh cột.

Biểu diễn bài toán. Chúng ta sử dụng nhiễn sắc thể có cấu trúc $(b^1,...,b^M)$, $b^k \in Z_+^m$ để biểu diễn các cá thể (các điểm) trong không gian tìm kiếm. Mỗi quần thể là một tập bao gồm một số cố định các cá thể.

Độ đo thích nghi. Với mỗi cá thể $s = (b^1,...,b^M)$ ta xác định mức độ thích nghi của cá thể, $f(s)$, bằng công thức sau:

$$f(s) = \frac{1}{\sum_{k=1}^{M} 1DCSP_G(m, L_k, l, b^k) c_k} \tag{20}$$

Toán tử hôn phối. Giả sử $s_1 = (b_1^1,...,b_1^M)$ và $s_2 = (b_2^1,...,b_2^M)$ là 2 cá thể bất kì, k là một số được lựa chọn ngẫu nhiên, $1 < k \leq m$. Từ hai cá thể trên ta tạo ra hai hậu duệ s_1' và s_2' với các vectơ cột tương ứng của chúng được xác định như sau:

$$b_1'^j[i] = b_1^j[i], i = 1,...,k-1; \qquad b_1'^j[i] = b_2^j[i], i = k,...,m; \quad j=1,...,M \tag{21}$$

$$b_2'^j[i] = b_2^j[i], i = 1,...,k-1; \qquad b_2'^j[i] = b_1^j[i], i = k,...,m; \quad j=1,...,M \tag{22}$$

Toán tử đột biến. Chọn ngẫn nhiên một bộ 3 các số nguyên (p,q,r), $1 \leq p, q \leq M$ và $1 \leq r \leq m$, với xác suất:

$$p_0 = 1/mM^2. \tag{23}$$

Toán tử đột biến tác động lên cá thể $s = (b^1,...,b^M)$ để tạo nên cá thể $s_1 = (b_1^1,...,b_1^M)$ với bộ *(p,q,r)* đã chọn như sau:

$$b_1^i = b^i \quad \text{khi} \quad i \neq p \,\&\, i \neq q \quad i=1,\ldots,M \tag{24}$$

$$b_1^p[j] = b^p[j],\ b_1^q[j] = b^q[j] \ \text{khi}\ j=1,\ldots,m \ \text{và}\ j \neq r \tag{25}$$

$$b_1^p[r] = b^p[r] - 1,\ b_1^q[r] = b^q[r] + 1 \quad \text{nếu}\ b^p[r] > 0 \tag{26}$$

$$b_1^p[r] = b^p[r],\ b_1^q[r] = b^q[r] \quad \text{nếu}\ b^p[r] = 0 \tag{27}$$

Toán tử chọn lọc. Toán tử chọn lọc được xác định theo luật tỉ lệ thuận với mức độ thích nghi:

$$p_s = \frac{f(s)}{\sum_{s \in G} f(s)} \tag{28}$$

trong đó s là cá thể và G là quần thể đang xem xét có chứa s.

Định lí 4. [1] *Giả sử hai cá thể cha-mẹ là các phân hoạch của cùng một vectơ. Khi đó các toán tử hôn phối và toán tử đột biến xác định như trên sẽ bảo đảm các hậu duệ cũng là những phân hoạch của vectơ đó.*

Dựa trên các Định lí 1,2,3,4 chúng ta có thể xây dựng thuật toán lai ghép như sau:

Thuật toán GA-CG

Input: ***m, M , l, L, b, c***

Output: Nghiệm tối ưu của bài toán $1DMCSP(m, M, l, b, L, c)$ và các phương án cắt tương ứng với nghiệm tối ưu đó

Step 0. Khởi tạo quần thể gồm K cá thể $G_0 = \{s_1^0, \ldots, s_K^0\}$. Việc khởi tạo này có thể thực hiện dễ dàng bằng việc tạo ra K phân hoạch khác nhau của b, mỗi phân hoạch sẽ tương ứng với một đối tượng của quần thể. Các cá thể được biểu diễn như sau:

$$s_i^0 = \left(b_{i1}^0, \ldots, b_{iM}^0\right),\ b_{ij}^0 \in Z_+^m\ ,\ \sum_{j=1}^{M} b_{ij}^0 = b\ ,\ i=1,\ldots,K$$

Step 1. Giải các bài toán $1DCSP_G(m, L_k, l, b_{ik}^t)$ bằng phương pháp tạo sinh cột , $i=1,\ldots,K$, $k=1,\ldots,M$, t là thứ tự bước lặp (thứ tự của quần thể). Tính mức độ thích nghi $f(s_i^t)$ cho từng cá thể của G_t theo (20).

Step 2. Lựa chọn các cha-mẹ trong G_t theo mức độ thích nghi để ghép cặp theo toán tử hôn phối (21)-(22) để tạo nên tập các hậu thế $G_t^{'}$ với K_1 phần tử

Step 3. Tác động toán tử đột biến (24)-(27) vào $G_t \cup G_t^{'}$ để nhận được $G_t^{''}$.

Step 4. Thực hiện tính toán giống như trong Step1 cho các cá thể của $G_t^{''}$.

Step 5. Áp dụng toán tử chọn lọc (28) lên $G_t \cup G_t^{''}$ để chọn ra K cá thể có mức độ thích nghi lớn nhất tạo quần thể mới G_{t+1}.

Step 6. Nếu điều kiện dừng chưa thỏa mãn quay lại Step 2. Ngược lai thuật toán dừng và cho nghiệm tối ưu cùng tập các phương án cắt tương ứng với nghiệm tối ưu.

Định lí 5. [1] *Thuật toán GA-CG đạt được nghiệm tối ưu toàn cục trong khoảng thời gian hữu hạn với xác suất 1 không phụ thuộc vào quần thể khởi đầu.*

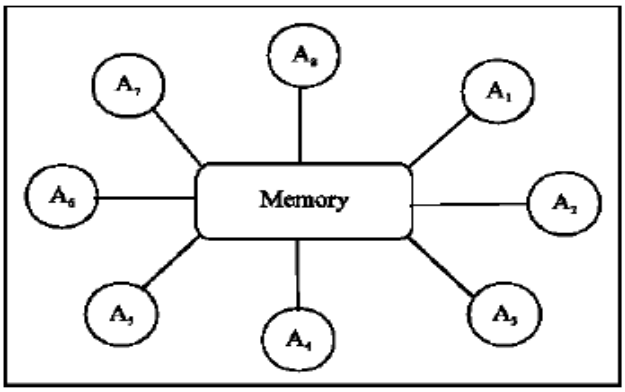
Các kết quả lí thuyết trong mục này sẽ được dùng làm cơ sở cho việc xây dựng hệ thống đa tác tử với tên gọi GMAS-1DMCSP được trình bày trong mục tiếp theo.

3. HỆ THỐNG GMAS-1DMCSP

Các Định lí 2 và 3 phản ánh mối quan hệ ngữ nghĩa giữa bài toán $1DMCSP(m,M,l,b,L,c)$ và bài toán $1DCSP_G(m,L_k,l,b^k)$. Việc giải bài toán $1DMCSP(m,M,l,b,L,c)$ theo thuật toán GA-CG chính là việc áp dụng thuật toán gen trên không gian tìm kiếm được tạo thành từ các tổ hợp nghiệm của các bài toán $1DCSP_G(m,L_k,l,b^k)$ với các b^k thỏa mãn ràng buộc $\sum_{k=1}^{M} b^k = b$. Từ quan sát này, chúng ta có thể xây dựng một hệ thống đa tác tử theo mô hình A-Team để giải bài toán $1DMCSP(m,M,l,b,L,c)$ với hai kiểu tác tử chính. Kiểu tác tử đầu tiên thực hiện thuật toán GA-CG và đóng vai trò khởi tạo, quản lí, lựa chọn nghiệm của bài toán $1DMCSP(m,M,l,b,L,c)$ trên cơ sở nghiệm của các bài toán thành phần $1DCSP_G(m,L_k,l,b^k)$. Kiểu tác tử thứ hai nhận nhiệm vụ giải bài toán $1DCSP_G(m,L_k,l,b^k)$ bằng kĩ thuật tạo sinh cột. Ngoài ra hệ thống cũng bao gồm một số tác tử đặc biệt cung cấp các dịnh vụ khác như quản lí tài nguyên, quản lí và báo cáo công việc... Các tác tử hoạt động phân tán, song song, không đồng bộ và có thể di trú một cách linh hoạt phụ thuộc vào tài nguyên được cấp phát. Hệ thống được cài đặt trên nền tảng JADE – một phần mềm nguồn mở hỗ trợ thiết kế và khai thác các hệ thống đa tác tử thành công nhất cho tới nay. Sau đâu sẽ là mô tả tóm tắt thiết kế của hệ thống.

3.1. Kiến trúc A-Team

Talukdar và cộng sự đã đề xuất một kiến trúc cơ sở cho phép xây dựng các hệ thống trong đó một nhóm các tác tử tự trị hoạt động một cách song song, không đồng bộ trong khuôn khổ cộng tác với nhau để giải các bài toán tối ưu với tên gọi **A-Teams** (Asynchronous Teams) [8, 9]. Trong kiến trúc cơ sở này, mỗi tác tử sẽ đảm trách một số kĩ năng nào đó trong việc giải quyết bài toán toàn cục và hoạt động hoàn toàn độc lập với các tác tử khác. Mỗi tác tử tự lựa chọn nghiệm từ quần thể nghiệm được lưu trữ trong bộ nhớ chung, thực hiện một số thao tác theo kĩ năng riêng của mình trên nghiệm đã chọn và trả lại kết quả đạt được vào bộ nhớ chung. Sự phối hợp giữa các tác tử được thực hiện thông qua việc chia sẻ các nghiệm trong bộ nhớ chung. Quần thể nghiệm được kiểm soát bởi một số tác tử chuyên trách (Destroyer agents) có nhiệm vụ đánh giá nghiệm theo một số tiêu chuẩn nào đó và đồng thời loại bỏ những nghiệm không mong muốn. Cách tổ chức như vậy cho phép các tác tử được liên kết lỏng lẻo có thể tham gia hoặc rời bỏ hệ thống một cách mềm dẻo, có thể khu trú phân tán một cách tự do trên các tài nguyên tính toán và không cần trao đổi thông tin trực tiếp với các tác tử khác. Đề xuất này tỏ ra phù hợp với nhiều ứng dụng phức tạp trong thực tế. Kiến trúc điển hình của A-Team được cho trong hình 1.



Hình 1. Kiến trúc của A-Team

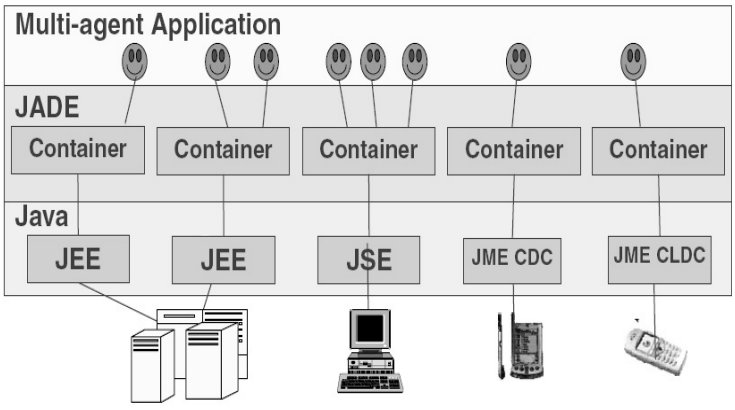
3.2. Nền tảng JADE

JADE là một framework phần mềm khá phổ biến với các tiện ích dùng để phát triển các hệ thống đa tác tử thông minh và được sử dụng rộng rãi như một công cụ hỗ trợ các hoạt động nghiên cứu cũng như phát triển ứng dụng thực tế [23]. Sản phẩm này được xây dựng dựa hoàn toàn vào các đặc tả của FIPA vì vậy bản thân JADE thường được đồng nhất với các đặc tả này.

Các ứng dụng xây dựng trên JADE được tổ chức như một tập hợp các thành phần (components) thuộc hai loại :

- Các tác tử - đây là các phần tử ngang hàng có tính tự trị và truyền thông với nhau bằng cách chuyển thông điệp không đồng bộ

- Các dịnh vụ (services) – là các thành phần không tự trị có thể chạy trên một hoặc nhiều node mạng theo cơ chế phối hợp và việc vận hành chúng có thể được kích hoạt bởi các tác tử.



Hình 2. Kiến trúc của một hệ đa tác tử JADE

JADE bao gồm các thư viện (Java classes) hỗ trợ cho việc thiết kế, cài đặt các tác tử và một môi trường thực thi cung cấp các dịch vụ cơ bản được kích hoạt trước khi các tác tử hoạt động. Mỗi bản thực thi của JADE được gọi là container - nơi khu trú của các tác tử. Tập tất cả các containers tạo thành một nền tảng (platform) và cung cấp một tầng thuần nhất che đi sự phức tạp và đa dạng của các tầng dưới (phần cứng, hệ điều hành, kiểu mạng lưới, JVM) khi thiết kế, cài đặt và khai thác các tác tử. Điều này giúp việc phát triển ứng dụng tác tử trở nên thuận lợi và tự nhiên hơn. Hình 2 thể hiện kiến trúc của một hệ thống đa tác tử xây dựng trên JADE và được triển khai trên tập các node tính toán không thuần nhất.

Mỗi tác tử xây dựng trên JADE được nhận biết bởi một định danh (identifier - ID) duy nhất và cung cấp một tập các dịch vụ. Các tác tử có thể đăng ký và sửa đổi dịch vụ của mình hoặc tìm kiếm tác tử cung cấp các dịch vụ mong muốn. Chúng có thể tự kiểm soát vòng đời của bản thân và đặc biết có thể trao đổi thông tin với các tác tử khác.

Các tác tử truyền thông tin bằng việc trao đổi thông điệp – mô hình được sử dụng một cách rộng rãi trong mô hình hóa tương tác phân tán có liên kết lỏng lẻo giữa các thực thể không thuần nhất. Để thực hiện việc truyền thông, mỗi tác tử chỉ cần gửi thông điệp tới định danh của tác tử nhận. Việc truyền thông theo định danh như vậy tạo ra nhiều lợi ích. Thứ nhất nó không đòi hỏi mối liên hệ tham chiếu tới đối tượng đích (Destination object reference) và không có sự phụ thuộc thời gian giữa bên gửi và bên nhận. Bên gửi và bên nhận có thể xuất hiện tại những thời điểm khác nhau. Bên nhận thậm chí có thể chưa tồn tại hoặc không được biết bởi bên gửi và bên gửi có thể dùng tính chất nào đó mà bên nhận phải thỏa mãn để xác định đích đến của thông điệp. Thứ hai, do các tác tử xác định các tác tử khác bằng tên nên mọi thay đổi các tham chiếu tới đối tượng (object reference) trong thời điểm thực thi là hoàn toàn trong suốt đối với ứng dụng.

Cấu trúc thông điệp trong JADE hoàn toàn tuân thủ ngôn ngữ truyền thông tác tử ACL (Agent Communication Language) do FIPA định nghĩa. Thông điệp bao gồm các trường (fields) để hỗ trợ các tương tác phức tạp cũng như đa hội thoại song song (multiple parallel conversations). JADE cũng hỗ trợ việc chuyển đổi tự động các thông điệp sang các định dạng khác như XML, RDF, Java Object và ngược lại nhằm phù hợp với những yêu cầu xử lí nội dung khác nhau.

3.3. Mô tả tóm tắt hệ thống GMAS-1DMCSP

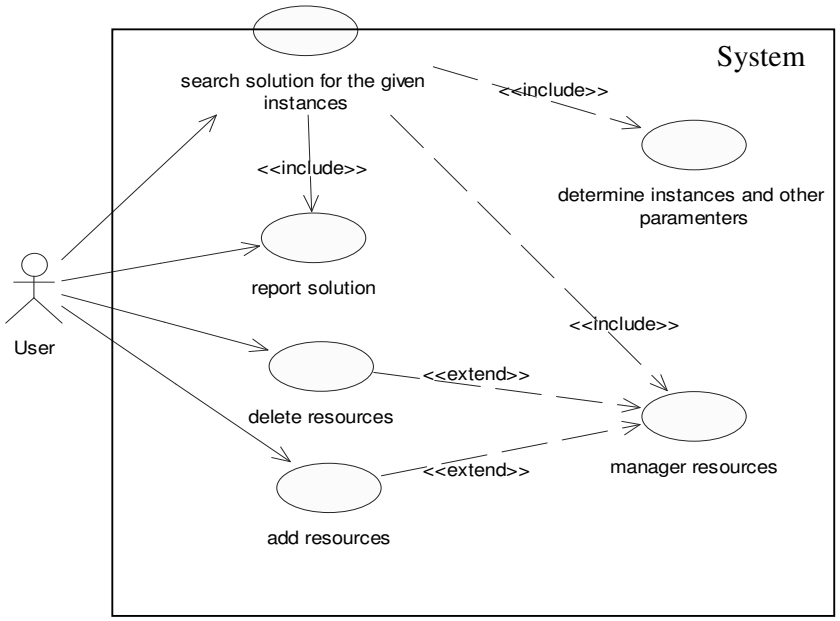
GMAS-1DMCSP là hệ thống đa tác tử cài đặt thuật toán GA-CG được trình bày trong mục 2 để giải quyết các bài toán cắt vật tư một chiều với kiến trúc A-Team trên nền JADE. Hệ thống có các đặc trưng sau đây :

- Hệ thống có thể giải quyết nhiều bài toán cắt vật tư khác nhau tại cùng một thời điểm

- Quá trình giải từng bài toán có thể được thực hiện trên nhiều máy tính. Người dùng có thể tự do kết nối hoặc loại bỏ một máy tính bất kì khỏi hệ thống. Trong trường hợp này hệ thống tự thích nghi với những thay đổi bằng việc chỉ thị cho các tác tử đang làm việc trong hệ thống di trú tới nơi cho phép.

- Hệ thống thực hiện việc giải các bài toán theo mẻ (batch mode). Các bài toán mới sẽ được lưu trữ và giải quyết khi hệ thống có đủ tài nguyên cho phép thực hiện giải bài toán.

Việc tương tác giữa người dùng với hệ thống trong quá trình giải bài toán được thể hiện bằng biểu đồ trong hình 3.



Hình 3. Biểu đồ tương tác giữa người dùng và hệ thống GMAS-1DMCSP

Chức năng chính của hệ thống GMAS-1DMCSP là tổ chức và thực hiện việc tìm kiếm nghiệm cho bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô với các bước như sau:

Đối với từng bài toán cụ thể

- Tạo sinh quần thể nghiệm khởi đầu và lưu chúng trong bộ nhớ chung (commom memory)

- Cải thiện chất lượng quần thể nghiệm bằng các tác tử có nhiệm vụ giải các bài toán $1DMCSP(m,M,l,b,L,c)$, $1DCSP_G(m,L_k,l,b^k)$ và lưu trữ quần thể nghiệm đã được cải thiện vào bộ nhớ chung. Quá trình cải thiện chất lượng này lặp lại cho tới khi điều kiện dừng được thỏa mãn.

Trong hệ thống, việc lưu trữ thông tin trong bộ nhớ chung được tổ chức như sau. Mỗi bài toán 1DMCSP cụ thể được xác định bởi một định danh duy nhất do người dùng đặt ra và được cấp phát một vùng nhớ trong bộ nhớ chung có cấu trúc gồm 3 miền.

Miền đầu tiên, Par, chứa các tham số đặc trưng của bài toán: *m, M, l, b, L, c* .

Miền thứ hai, Feasible Solution – FS, chứa quần thể nghiệm chấp nhận được gồm *K* nghiệm (*K* phân hoạch nào đó của vectơ *b,* với *K* là kích thước quần thể nghiện trong GA-CG) của bài toán $1DMCSP(m,M,l,b,L,c)$. Mỗi nghiệm chấp nhận được có cấu trúc bao gồm một biến trạng thái nghiệm, một phân hoạch của vectơ kế hoạch *b*, một vectơ trạng thái của các bài toán $1DCSP_G(m,L_k,l,b^k)$ tương ứng với từng lớp của phân hoạch, các vectơ nghiệm tối ưu *x* và các ma trận phương án cắt tương ứng của từng bài toán $1DCSP_G(m,L_k,l,b^k)$. Mỗi biến trạng thái có thể nhận 3 giá trị: chưa được xử lí (Np), đang xử lí (P) và hoàn thành (F) tương ứng

với việc bài toán chưa được xem xét, bài toán đang được xử lí và bài toán đã được giải quyết xong. Ban đầu tất cả các biến trạng thái đều nhận giá trị Np. Các biến trạng thái có mối quan hệ phân cấp. Biến trạng thái ở mức trên sẽ có giá trị Np nếu tất cả các biễn trạng thái ở mức dưới đều có giá trị Np; có giá trị P nếu ít nhất một biến trạng thái ở mức dưới có giá trị P hoặc F và ít nhất một trong các biến còn lại không có giá trị F; khi tất cả các biến trạng thái ở mức dưới có giá trị F thì biến trạng thái trực tiếp ở mức trên sẽ được gán giá trì F.

Miền thứ ba, Offspring, chứa các nghiệm chấp nhận được của bài toán $1DMCSP(m,M,l,b,L,c)$ được sinh ra từ các nghiệm chấp nhận được trong miền FS bằng tác động của các toán tử gen (21) - (22), (24) - (27) và (28). Cấu trúc bộ nhớ chung có thể được mô tả dưới định dạng XML như sau:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE boundle SYSTEM "boundle.dtd">
<boundle>
<task>
<problemID>INSTANCE-1</problemID>
<Par>
<dataclass> Data</dataclass>
</Par>
<state>STATE</state>
<FS>data</FS>
<Offspring>data</Offspring>
</task>
</boundle>
```

Hệ thống bao gồm 4 loại tác tử gồm TaskManager, ResourceManager, 1DMCSP-Solvers và 1DCSP-Solvers được phân bố động trên các node tính toán của mạng LAN và hoạt động hoàn tòan độc lập với nhau theo cơ chế không đồng bộ. Số lượng các tác tử của hai loại 1DMCSP-Solvers và 1DCSP-Solvers được sinh ra phụ thuộc số lượng bài toán cần giải quyết và tài nguyên của hệ thống. Các tác tử mỗi loại đảm trách các chức năng cụ thể như sau.

TaskManager

- Khởi tạo và quản lí trạng thái của các bài toán
- Trả kết quả cho người dùng khi bài toán được giải xong

ResourceManager

- Quản lí platform, bộ nhớ chung

1DMCSP-Solvers

- Mỗi bài toán sẽ do một **1DMCSP-Solver** chịu tránh nhiệm xử lí. Tác tử này phối hợp với các tác tử 1DCSP-Solvers thực hiện thuật toán GA-CG để giải các bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô

1DCSP-Solvers

- Thực hiện giải thuật tạo sinh cột của Gilmore và Gomory để giải bài toán cắt vật tư một chiều với một loại vật liệu thô. Tập các tác tử 1DCSP-Solvers được sử dụng chung và tham gia giải quyết mọi bài toán được khởi tạo trong bộ nhớ.

Hoạt động của hệ thống có thể được mô tả tóm tắt như sau.

Khi khởi động hệ thống, người dùng thông qua TaskManager chỉ ra các tài nguyên có thể sử dụng (các node tính toán trên mạng LAN). TaskManager khởi tạo một tác tử ResourceManager và truyền các thông tin này cho ResourceManager. ResourceManager sẽ khởi tạo platform, tổ chức một bộ nhớ chung, tạo sinh một số bản copy của tác tử 1DCSP-Solver (số bản copy phụ thuộc vào độ lớn của các container) đồng thời phân bố chúng trên các container khác nhau.

Ban đầu, dữ liệu của bài toán được người dùng soạn thảo dưới định dạng XML như trên và được lưu thành các file trong một folder ở bộ nhớ ngoài. Tại mỗi thời điểm bắt đầu hoạt động của hệ thống, người dùng sẽ cung cấp đường dẫn folder dữ liệu này và tác tử TaskManager sẽ đọc các tham số của từng bài toán, tính toán không gian nhớ cần thiết để lưu trữ dữ liệu của bài toán tại bộ nhớ chung và yêu cầu tác tử ResourceManager cung cấp bộ nhớ có kích thước phù hợp. ResourceManager nhận yêu cầu từ TaskManager, kiểm tra không gian nhớ còn chưa sử dụng của bộ nhớ chung. Nếu không gian nhớ chưa sử dụng của bộ nhớ chung đủ để lưu trữ dữ liệu của bài toán mới, ResourceManager sẽ thông báo hoạt động thành công, và cung cấp địa chỉ đầu của không gian nhớ còn chưa sử dụng cho TaskManager. TaskManager sẽ lưu các dữ liệu của bài toán vào các miền tương ứng và gửi thông báo yêu cầu ResouceManager tạo sinh một một bản copy tác tử 1DMCSP-Solver dành riêng để thực hiện giải bài toán mới bằng việc cung cấp cho tác tử này problemID của bài toán do TaskManager khởi tạo.

Trong trường hợp hệ thống đang hoạt động, nếu TaskManager nhận được yêu cầu dừng từ người dùng nó sẽ thay các đổi các biến trạng thái đang có giá trị P của các bài toán $1DCSP_G(m, L_k, l, b^k)$ thành giá trị Np và chuyển đổi toàn bộ dữ liệu trong bộ nhớ chung thành định dạng XML như trên và lưu ra thiết bị ngoại vi. Hành vi trên của TaskManager cũng được đặt lịch định kì (thời gian do người dùng lựa chọn) nếu không có tác động của người dùng. Cách tổ chức như vậy bảo đảm cho việc hệ thống có thể khôi phục lại trạng thái giải quyết bài toán gần nhất khi có sự cố xảy ra.

Trong quá trình hệ thống đang hoạt động, người dùng có thể chỉ thị thêm hoặc bớt tài nguyên. Khi đó TaskManager sẽ yêu cầu ReourceManager phân bố lại các tác tử đang hoạt động của hệ thống bằng việc chỉ thị cho các tác tử di trú từ một container sang một container khác (các tác tử trong hệ thống là các tác tử di động -Mobile Agent) để bảo đảm hệ thống hoạt động một cách hiệu quả.

Khi có thông báo thành công từ tác tử 1DMCSP-Solver nào đó, TaskManager trả kết quả bài toán tương ứng cho người dùng và thông báo yêu cầu ResourceManager giải phóng miền nhớ trong bộ nhớ chung ứng với bài toán đã được xử lí xong.

Phụ thuộc vào không gian nhớ chưa sử dụng trong bộ nhớ chung do mình quản lí, ResourceManager ước lượng khả năng phân phối không gian nhớ cho bài toán mới. Nếu không gian nhớ chưa sử dụng vượt qua giá trị trung bình không gian nhớ của tất cả các bài toán đang lưu trữ trong bộ nhớ chung, ResourceManager gửi thông báo cho TaskManager. Trong trường hợp này, TaskManager kiểm tra folder lưu trữ dữ liệu của người dùng. Nếu có bài toán chưa được khởi động TaskManager sẽ hoạt động như tại thời điểm xuất phát của hệ thống.

Việc giải bài toán được thực hiện chủ yếu nhờ hai loại tác tử : 1DCSP-Solvers và 1DMCSP-Solvers.

Mỗi tác tử loại 1DCSP-Solver có nhiệm vụ giải bài toán cắt vật tư một chiều với một loại vật liệu thô bằng kĩ thuật tạo sinh cột của Gilmore và Gomory. Các tác tử này định kì theo thời gian duyệt nội dung các miền FS và Offspring của bộ nhớ chung để tìm các thể hiện chưa được xử lí của bài toán $1DCSP_G(m, L_k, l, b^k)$ thông qua định danh của bài toán, số thứ tự của nghiệm chấp nhận được, số thứ tự của bài toán $1DCSP_G(m, L_k, l, b^k)$ trong thành phần của nghiệm chấp nhận được và giá trị của biến trạng thái gắn với nó. Nếu có bài toán chưa được xử lí trong bộ nhớ chung (giá trị biến trạng thái là Np), 1DCSP-Solver sẽ đọc dữ liệu của bài toán, gắn giá trị của biến trạng thái thành P, giải bài toán và cập nhật kết quả vào bộ nhớ chung cùng với việc gán giá trị biến trạng thái của bài toán thành F. Quá trình được lặp lại khi chưa nhận được các chỉ thị khác từ TaskManager.

Mỗi tác tử loại 1DMCSP-Solver giải bải toán do mình đảm trách theo các bước cụ thể sau :

- Tạo sinh quần thể nghiệm chấp nhận được ban đầu cho bài toán 1DMCSP theo Step 0 của thuật toán GA-CG. Lưu các nghiệm chấp nhận được vào miền FS của bộ nhớ chung dành cho bài toán và xóa miền Offspring tương ứng.

- Đối với bài toán cụ thể được giao đảm trách, định kì kiểm tra biến trạng thái của các nghiệm chấp nhận được của nó. Nếu mọi nghiệm chấp nhận được trong miền FS dành cho bài toán đã được xử lí xong (biến trạng thái của mọi nghiệm chấp nhận được có giá trị F (đồng nghĩa với việc tất cả các bài toán $1DCSP_G(m, L_k, l, b^k)$ trong mọi nghiệm chấp nhận được đã được giải thành công bởi các tác tử loại 1DCSP-Solver) và miền offspring tương ứng là rỗng, 1DMCSP-Solver sẽ thực hiện các Step 2 và Step 3 của thuật toán GA-CG để nhận được tập nghiệm chấp nhận được $G_t^{''}$ và lưu tập nghiệm này vào miền Offspring của bộ nhớ chung đồng thời gán giá trị Np cho các biến trạng thái của từng nghiệm cũng như của các bài toán $1DCSP_G(m, L_k, l, b^k)$ thành phần. Trong trường hợp tất cả các nghiệm chấp nhận được trong cả hai miền FS và Offspring đều có giá trị biến trạng thái là F, 1DMCSP-Solver sẽ thực hiện Step 5 của GA-CG bằng cách thay thế các nghiệm chấp nhận được có độ thích nghi thấp trong FS bởi các nghiệm chấp nhận được có độ thích nghi cao hơn trong miền Offspring. Sau khi đã thực hiện xong việc thay thế, nội dung miền nhớ Offspring được xóa. Quá trình này lặp lại tới khi điều kiện kết thúc của bài toán được thỏa mãn và khi đó gán giá trị F cho biến trạng thái của bài toán. Sau khi hoàn thành việc giải bài toán, tác tử này thông báo thành công cho TaskManager, ResourceManager và tự hủy. ResouceManager cập nhật thông tin platform.

Đặc tả chức năng bằng giả mã của từng loại tác tử được đưa ra trong phần Phụ lục của bài này. Hoạt động của các tác tử trong hệ thống được thể hiện trong hình 4.

Định lí 6. *Hành vi của các tác tử thuộc hai loại 1DMCSP-Solver và* $1DCSP_G(m, L_k, l, b^k)$ *trong hệ thống bảo đảm cho kết quả tối ưu của từng bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô và hệ thống có khả năng giải quyết đồng thời nhiều bài toán 1DMCSP*

*Chứng minh.*Thật vậy, với các hành vi và quá trình tương tác của các loại tác tử như được đặc tả trong đặc tả chức năng, chúng ta dễ ràng thấy rằng:

- Các tác tử hai loại TaskManager, ResourceManager không tham gia trực tiếp vào quá trình giải các bài toán cắt vật tư. Bởi vậy hoạt động của chúng chỉ có tác dụng tăng cường hiệu quả

của hệ thống trong việc quản lí tài nguyên và cho phép hệ thống có khả năng giải quyết đồng thời nhiều bài toán cắt vật tư khác nhau.

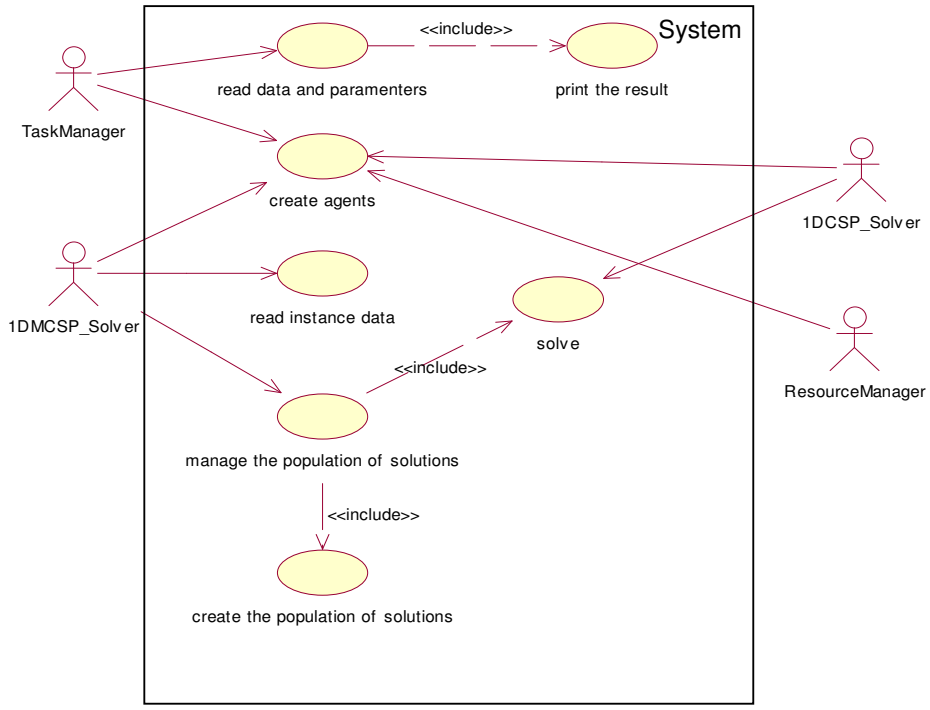
- Các tác tử 1DCSP-Solver thực hiện việc giải các bài toán $1DCSP_G(m, L_k, l, b^k)$ nằm trong Step 1 và Step 4 của thuật toán GA-CG

- Các tác tử 1DMCSP-Solver thực hiện các Step 0, Step 2, Step 3, Step 5 và Step 6 của thuật toán GA-CG.

-Nhờ cấu trúc miền bộ nhớ chung giành cho từng bài, sự phối hợp của hai loại tác tử được ghi nhận bởi việc cập các biến trạng thái tuân theo đúng logic tính toán của thuật toán GA-CG.

- Trong trường hợp có sự cố xảy ra, hệ thống có khả năng khôi phục lại trạng thái của bài toán từ thời điểm cuối cùng khi TaskManager thành công trong việc chuyển đổi dữ liệu của bộ nhớ chung thành định dạng XML và ghi vào thiết bị ngoại vi. Các tác tử sẽ thực hiện công việc tiếp tục từ thời điểm này.

Từ đó, tính đúng đắn của hệ thống trong việc giải quyết từng bài toán đơn lẻ được suy ra trực tiếp từ Đinh lí 5. Điều này cũng khẳng định tính đúng đắn của toàn bộ hệ thống khi giải quyết đồng thời nhiều bài toán nhờ sự phối hợp của các tác tử TaskManager và ResouceManager với cấu trúc miền nhớ dành cho từng bài toán.



Hình 4. Hoạt động của các tác tử trong hệ thống

4. ĐÁNH GIÁ HỆ THỐNG TRONG THỰC TẾ

Mối liên hệ ngữ nghĩa của bài toán cắt vật tư 1 chiều cho nhiều loại vật tư với bài toán cắt vật tư 1 chiều cho một loại vật tư được thể hiện trong các Định lí 1,2,3 là nền tảng lí thuyết cho việc song song và phân tán hóa thuật toán GA-CG để tăng tính hiệu quả về thời gian của thuật toán. Trong thực tế cài đặt, nhóm tác giả đã cài đặt thuật toán dưới dạng một hệ thống đa tác tử (Multi-agent System) GMAS-1DMCSP. Hệ thống gồm 2 lớp tác tử chính. Lớp trên gồm 1 agent thực hiện thuật toán gen có nhiệm vụ lập kế hoạch (tạo ra các phân hoạch của vectơ kế hoạch), điều phối các agent ở lớp dưới và đồng bộ hóa kết quả (tính độ đo thích nghi, thực hiện các toán tử gen…). Lớp dưới gồm các agent thuần nhất (các agent đều cùng thực hiện thuật toán tạo sinh cột để giải bài toán cắt vật tư 1 chiều cho 1 loại vật liệu thô). Các agent của lớp dưới là các agent di động (Mobile agent) được agent lớp trên tạo sinh khi có nhu cầu và có thể khu trú tại những node tính toán bất kì trên mạng cục bộ có đủ tài nguyên. Đây cũng là điều vượt trội của thuật toán của chúng tôi so với các thuật toán mà các tác giả khác đề xuất. Việc cài đặt như vậy đã tận dụng được sức mạnh tính toán của mạng LAN và góp phần giảm đáng kể thời gian giải bài toán. Nếu bỏ qua vấn đề truyền thông trong mạng LAN, sự cải thiện về tốc độ tính toán của hệ thống GMAS-1DMCSP so với phiên bản cài đặt tập trung của thuật toán GA-CG được thể hiện bởi những ước lượng sau

Gọi thời gian tính toán tối đa để giải bài toán $1DCSP_G(m, L, l, b)$ bằng phương pháp tạo sinh cột là T_G. Tại mỗi vòng lặp của thuật toán GA-CG, chúng ta cần giải M bài toán $1DCSP_G(m, L_k, l, b^k)$ với M là số lượng các loại kích thước vật liệu thô. Như vậy nếu thuật toán đáp ứng được điều kiện dừng sau N bước lặp, ước lượng thời gian tính toán của GA-CG sẽ là

$$N \times M \times T_G . \tag{29}$$

Giả sử số lượng các tác tử 1DCSP-Solver có thể được tạo ra trong hệ thống GMAS-1DMCSP là K. Khi đó trong mỗi vòng lặp của 1DMCSP-Solver, M bài toán $1DCSP_G(m, L_k, l, b^k)$ sẽ được K tác tử 1DCSP-Solver chia nhau giải một cách độc lập bằng phương pháp tạo sinh cột. Như vậy thời gian đòi hỏi của mỗi vòng lặp trong 1DMCSP-Solver có thể xấp xỉ bởi công thức :

$$\left\lceil \frac{M}{K} \right\rceil \times T_G .$$

Nhận xét rằng để bảo đảm thỏa mãn điều kiện dừng, tác tử 1DMCSP-Solver của hệ thống GMAS-1DMCSP cũng có số bước lặp đúng như số bước lặp của GA-CG. Như vậy thời gian để đòi hỏi của hệ thống để giải bài toán xấp xỉ là :

$$N \times \left\lceil \frac{M}{K} \right\rceil \times T_G .$$

Trong thực tế triển khai tại nhà máy ống thép Việt-Đức thì số lượng các loại kích thước vật liệu thô M thường ít hơn 20 và mạng LAN của nhà máy được bố trí tập trung tại tòa nhà điều hành với 15 node mạng. Mỗi node mạng được phân phối một container và được liên kết theo kiến trúc của JADE. Số lượng các tác tử 1DCSP-Solver trong mỗi container phụ thuộc cấu hình của node và trên thực tế triển khai chúng tôi có thể tạo ra một số lượng khá lớn. Bởi vậy số lượng K các tác tử 1DCSP-Solver trên hệ thống lớn hơn M nhiều lần. Do đó:

$$N \times \left\lceil \frac{M}{K} \right\rceil \times T_G = N \times T_G \ . \qquad (30)$$

So sánh (29) với (30), ta thấy hệ thống GMAS-1DMCSP có thể nhanh gấp M lần so với phiên bản tập trung của GA-CG. Các ghi nhận thực tế cũng cho thấy việc truyền thông của hệ thống trên mạng LAN làm tăng không quá 2% thời gian ước lượng trong (30). Điều này cho thấy khi M càng lớn thì GMAS-1DMCSP càng thể hiện tính ưu việt so với GA-CG.

Hệ thống này đã được triển khai phục vụ sản xuất tại nhà máy ống thép Việt-Đức trong 4 năm qua. Kết quả cho thấy trên thực tế chưa gặp một đợt sản xuất nào có lượng phế thải lớn hơn 3% bề rộng của tấm vật tư có kích thước lớn nhất. Điều đó chỉ ra rằng thuật toán GA-CG được thiết kế như trên có tính chất tương tự như tính chất IRUP bằng chính các thẩm định thực tế, điều mà các tác giả khác không đề cập tới.

5. KẾT LUẬN

Bài báo trình bày tóm tắt việc thiết kế và cài đặt một hệ thống đa tác tử di động nhằm nâng cao hiệu quả việc giải các bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô trên cơ sở các kết quả phân tích lí thuyết của bài toán và thuật toán được trình bày trong [1].

Hệ thống được xây dựng theo kiến trúc A-team và được triển khai trên nền tảng JADE bằng ngôn ngữ Java.

Tính đúng đắn của hệ thống được chứng minh chặt trẽ bằng Định lí 6 phát biểu trong Mục 3 của bài này.

Hệ thống đã được triển khai và khai thác hiệu quả tại nhà máy ống thép Việt-Đức [1].

TAI LIỆU THAM KHảO

1. Phan Thị Hoài Phương, Lương Chi Mai, Nguyễn Văn Hùng - Một thuật toán lai ghép giải bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô, Tạp chí Tin học và Điều khiển học **25** (3) (2009) 214-230.
2. D. D. Corkill - Blackboard systems. AI Expert **6** (9) (1991) 40-47.
3. R. S. Engelmore and A. Morgan (Ed.) - Blackboard Systems, Addison-Wesley, 1988.
4. V. Jagannathan, R. Dodhiawala, and L. S. Baum (Ed.) - Blackboard Architectures and Applications, Academic Press, 1989.
5. A. Kazemi and M.H. Fazel Zarandi - An Agent-Based Framework for Building Decision Support System in Supply Chain Management, Journal of Applied Sciences **8** (7) (2008) 1125-1137.
6. Dariusz Barbucha and Piotr Je¸drzejowicz - An Agent-Based Approach to Vehicle Routing Problem, World Academy of Science, Engineering and Technology 26 2007
7. C. Chira, C.-M. Pintea, D. Dumitrescu - An Agent-Based Approach to Combinatorial Optimization, Int. J. of Computers, Communications & Control **III** (2008), Suppl. issue: Proceedings of ICCCC 2008. pp. 212-217.
8. S. Talukdar - Asynchronous Teams, Proceedings of The 4th International Symposium on Expert Systems Applications to Power Systems, LaTrobe University, Melbourne, Australia, 1993.

9. S. Talukdar et all. - Asynchronous Teams*:* Co-operation Schemas for autonomous, Computer-Based agents, Technical Report EDRC 18-59-96, Engineering Design Research Center, Carnegie Mellon University, 1996.
10. G. Belov and G. Scheithauer - A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths, European Journal of Operational Research, Special issue on cutting and packing **141** (2) (2002) 274-294.
11. G. Belov and G. Scheithauer - A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, Technical report, Dresden University, 2003, URL: www.math.tu-dresden.de/˜capad.
12. G. Belov and G. Scheithauer - Setup and open stacks minimization in one-dimensional stock cutting, Technical report, Dresden University, 2003.
13. Gleb Belov - Problems, Models and Algorithms in One- and Two-Dimensional Cutting, Dissertation. TU Dresden, 2004.
14. Gilmore P. C., R. E. Gomory - A linear programming approach to the cutting-stock problem, Oper. Res. **9** (1961) 849–859.
15. Gilmore P. C., R. E. Gomory - A linear programming approach to the cutting stock problem, Part II. Oper. Res. **11** (1963) 863–888.
16. J. Rietz and S. Dempe - Large Gaps in One-dimensional Cutting Stock Problems. Discrete Applied Mathematics **156** (10) (2008).
17. Sirinat Wongprakornkul - Round Down Technique for Solving an Integer Linear Programming, KKU Science Journal **36** (2008) 187-198
18. O. Holthaus - Decomposition approaches for solving the integer one dimensional cutting stock problem with different types of standard lengths, European Journal of Operational Research **141** (2002) 295-312.
19. Piotr Jędrzejowicz and Izabela Wierzbowska - JADE-Based A-Team Environment, Lecture Notes in Computer Science **3993** (2006) 719-726.
20. Gawinecki Maciej, Frackowiak Grzegorz - Multi-Agent Systems with JADE: A Guide with Extensive Study. Distributed Systems Online, IEEE **9** (3) (2008).
21. Charles V. Trappey et all. - The design of a JADE-based autonomous workflow management system for collaborative SoC design, Expert Systems with Applications **36** (2009) 2659–2669.
22. Fabio Bellifemine et all. - JADE: A software framework for developing multi-agent applications. Lessons learned, Information and Software Technology **50** (2008) 10-21.
23. F. Bellifemine, G. Caire, D. Greenwood - Developing multi-agent systems with JADE, Wiley Series in Agent Technology, ISBN 978-0-470-05747-6, February 2007.

PHỤ LỤC

Giả mã Đặc tả chức năng của các loại tác tử

A. **Giả mã đặc tả chức năng của tác tử** *TaskManager*

Start;
Receive "Resource information from User";
Create *ResourceManager*;
Send "Resource information" to *ResourceManager*;
Wait for "success platform preparation respond" from *ResourceManager*;
If (respond=failure) **Then**
{
Inform user about the failure;
Stop
};
Confirm "New batch?";
If (New Batch="Esc")
Then
{
Receive "link to immediate results of old batch";
Download « immediate results of old batch to common memory » ;
Create *1DMCSP-Solvers* corresponding to the unsolved problems
}
Else
Receive « link to new batch » ;
Set *Timer(*600s);
While "unsolved problem is available in the folder"
Do
{
Send Request "to allocate common memory space for new problem" to *ResourceManager*;
Wait for "success allocation respond" from *Resource Manager*;
If (respond=failure)
Then
While "the common memory is not empty"
Do

{

Repeat

Look for "solved problem in the common memory"

Until "solved problem found";

Print "The result of the solved problem";

Send Request "to free the common memory space for the solved problem" to *ResourceManager*

}

Else

{

Create 1DMCSP-Solver for New problem ;

Move New problem from the folder

} ;

If (*Timer expire* or KeyboardHook= « Shift »)

Then

{

Store « common memory content in secondary memory » ;

Set *Timer*(600s)

} ;

If (KeyboardHook= « Ctrl »)

Then

{

Receive « New Resource information from user » ;

Send request « To instruct the agents to move to new locations » to *ResourceManager*

}

} ;

While ("the common memory is not empty")

Do

{

Repeat

Look for "solved problem in common memory"

Until "solved problem found";

Print "The result of the solved problem"

};

Send Request "Stop Request" to *ResourceManager;*

Stop

B. **Giả mã đặc tả chức năng của tác tử** *ResourceManager*

Start;
Wait for "Resource information" from *TaskManager;*
Create platform and common memory;
Send Respond "Status of creating Platform and common memory" to TaskManager;
If (Status=failure) **Then Stop**;
Create *1DCSP-Solvers* ;
While (**True**)
Do
{
Wait for Request from *TaskManager;*
If (Request="to allocate common memory space for new problem")
Then
{
Check « fee space in common memory » ;
If (« free space is enouth for new problem »)
Then
Send Respond "success allocation respond" to *TaskManager*
Else
Send Respond "failure allocation respond" to *TaskManager*;
} ;
If (Request="to free the common memory space for the solved problem")
Then
{
Update "common memory management data";
Remove *1DMCSP-Solvers* corresponding to the solved problem from platform
};
If (Request=« Instruct the agents to move to new locations »)
Then
Reallocate « *1DMCSP-Solvers* and *1DCSP-Solvers to new resources » ;*
If (Request="Stop Request")
Then Stop
};

C. **Giả mã đặc tả chức năng của tác tử** *1DMCSP-Solver*

```
Start ;
Initiate « first feasible solution population in FS» ;
While (« Stop conditions are not satisfied »)
        Do
          {
           While (some feasible solution's Status Variable in FS and Offspring is not equal to F)
                    Do
                      {
                       Check  all Status variables of its 1DCSP subproblems ;
                       If  (all Status Variables of its 1DCSP subproblems  are F)
                           Then
                               Set  feasible solution's Status Variable=F
                       } ;
                 If  (Offspring is empty)
                    Then
                        {
                         Perform operations in Step 2, Step 3 and  of GA-CG ;
                         Store Results in Offspring ;
                         While (some feasible solution's Status Variable in Offspring is not equal to F)
                                  Do
                                    {
                                     Check  all Status variables of its 1DCSP subproblems ;
                                     If  (all Status Variables of its 1DCSP subproblems  are F)
                                       Then
                                            Set  feasible solution's Status Variable=F ;

                                     } ;
                         Perform Step 4 of GA-CG
                         }
                    Else
                        {
                         Peform Step 5 of GA-CG ;
                         Erase Offspring
                         }
                } ;
Set  Problem's Status Variable=F ;
Stop
```

D. **Giả mã đặc tả chức năng của tác tử** *1DCSP-Solver*

```
Start ;
While (TRUE)
    Do
      {
       Find « Unsolved 1DCSP subproblem in common memory » ;
       If  (« Unsolved 1DCSP subproblem found »)
           Then
                {
                 Set 1DCSP subproblem's Status Variable=P ;
                 Perform Step 1 of GA-CG  with unsolved 1DCSP subproblem ;
                  Store Results in common memory ;
                  Set 1DCSP subproblem's Status Variable=F
                  }

       }
```


SUMMARY

GMAS-1DMCSP: GENETIC MULTI-AGENT SYSTEM FOR SOLVING ONE-DIMENSIONAL CUTTING STOCK PROBLEM WITH MULTIPLE STOCK SIZES

Multi-agent systems (MASs) as an emerging sub-field of artificial intelligence concern with interaction of agents to solve a common problem. This paradigm has become more and more important in many aspects of computer science by introducing the issues of distributed intelligence and interaction. They represent a new way of analyzing, designing, and implementing complex software systems. This paper introduces a JADE-Based distributed Multi-agents system for solving One-Dimensional Cutting Stock Problem with Multiple Stock Sizes on the basis of the theoretical results presented in [1]. The system has effectively been operating in the industrial environment.

Địa chỉ: *Nhận bài ngày 25 tháng 7 năm 2009*

Phan Thị Hoài Phương,

Học viện Công nghệ Bưu chính viễn thông.

Lương Chi Mai,

Viện Công nghệ thông tin, Viện Khoa học và Công nghệ Việt Nam.