KIPS Tr. Software and Data Eng.
Vol.3, No.3 pp.119~124 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2014.3.3.119

Study for the Maximum Bipartite Subgraph Problem Using GRASP + Tabu Search

Keunhee Han† · Chansoo Kim††

ABSTRACT

Let $G = (V, E)$ be a graph. Maximum Bipartite Subgraph Problem is to convert a graph G into a bipartite graph by removing minimum number of edges. This problem belongs to NP-complete; hence, in this research, we are suggesting a new metaheuristic algorithm which combines Tabu search and GRASP.

Keywords : Graph, Bipartite Graph, Maximum Bipartite Subgraph Problem, Tabu Search, GRASP

Maximum Bipartite Subgraph 문제를 위한 GRASP + Tabu Search 알고리즘 연구

한 근 희† · 김 찬 수††

요 약

$G = (V, E)$ 를 그래프라 하자. Maximum Bipartite Subgraph 문제는 주어진 그래프 G로부터 최소 개수의 간선을 제거함으로써 G 를 이분그래프로 변환시키는 문제이며 결합 최적화 문제들 중 대표적인 문제들 중의 하나로 알려 져 있다. 본 문제는 NP-complete 계열에 포함되는 문제로서 본 연구에서는 Tabu Search 및 GRASP 등을 조합한 새로운 메타휴리스틱 알고리즘을 제시하고자 한다.

키워드 : 그래프, 이분 그래프, Maximum Bipartite Subgraph 문제, Tabu Search, GRASP

1. 서 론

$G = (V, E)$ 를 정점 집합 $V = \{1, 2, ..., n\}$ 및 간선 집합 $E \subseteq V \times V$로 구성되는 단순 무 방향 그래프라 하고 $|V| = n$ 및 $|E| = m$ 이라 하자. 간선 $e \in E$ 에 대하여 e 의 양 끝점을 u및 v라 하면 $e = uv$로 표기한다. $X \subseteq V$ 라 하고 $E(X) = \{uv | u, v \in E\}$ 라 하자. 만일 $u, v \in X$에 대하여 $uv \notin E$라면 X 를 *independent set*이라 하며, 만일 V가 두 개의 independent set 인 X 및 Y ($X \neq \emptyset$, $Y \neq \emptyset$) 로 분할될 수 있다면 G를 이분 그래프 (*Bipartite Graph*) 라 하며 $G = (X \cup Y, E)$로 표기된다. 이때 X 및 Y를 이분 그래프인 G 의 *partite set* 이라 한다.

만일 G가 이분 그래프가 아니라면 G로부터 적절한 간선들을 제거함으로써 G를 이분 그래프로 변환할 수 있다. 주어진 그래프 $G = (V, E)$로부터 $V = X \cap Y$ 및 $X \cup Y = \emptyset$로 분할한 후 $E(X)$ 및 $E(Y)$를 제거하는 것을 G의 *bipartition* 이라 하고 이때 $S = \{uv \in E | u \in X, v \in Y\}$를 G 의 *edge cut* 그리고, $|S|$를 *cutsize* 라 하자. *Maximum Bipartite Graph Problem* (*MBSP*) 는 주어진 그래프 G를 bipartition 을 통하여 이분그래프로 변환하는 것이며 이때 모든 가능한 bipartition들 중 cutsize 의 크기를 최대화시키는 것을 목적으로 하는 최적화 문제이다. 이는 다시 표현하면 MBSP 는 G로부터 최소 개수의 간선을 제거함으로써 G를 이분그래프로 변환시키는 문제이다.

MBSP가 NP-complete라는 것은 [1, 2] 에 의하여 증명이 되었으며, 본 문제는 조합 최적화 문제(Combinatorial Optimization Problem)들 중 대표적인 문제 중의 하나로 알려 져 있다 [3]. 본 문제는 NP-complete이므로 일반 그래프에 대하여 정확한 해를 계산하는 것은 $O(2^{n-1})$-time 이 요구되므로, n 이 증가함에 따라 정확한 해를 계산하는 것은 현실적으로 불가능하다. 그러나, 만일 그래프 $G = (V, E)$의 최대 차수가 3 이하이면서 triangle-free graph 라면 cutsize의 크기가 최소

† 종신회원 : 공주대학교 응용수학과 교수
†† 정 회 원 : 공주대학교 응용수학과 교수
논문접수 : 2013년 11월 18일
수 정 일 : 1차 2014년 1월 7일
심사완료 : 2014년 1월 9일
* Corresponding Author : Chansoo Kim(chanskim@kongju.ac.kr)

(4/5)|E|인 bipartition 을 계산하는 polynomial time algorithm 이 [4] 에서 제시되었으며, [5] 및 [6] 에서는 일반그래프에 대하여 본 문제를 해결하기 위한 Neural Network algorithm 들이 제시되었다.

Tabu Search (TS) 및 *Greedy Randomized Adaptive Search Procedure* (GRASP)는 각각 Glover (1986년) [7] 및 Resende (1989년) [8]에 의하여 최초로 제안된 알고리즘으로서 Genetic Algorithm, Simulated Annealing Algorithm 등과 함께 조합 최적화 문제들의 근사값을 계산하는데 널리 사용되고 있는 알고리즘이다. 본 연구에서는 MBSP를 위하여 GRASP 및 TS의 장점들을 선택적으로 적용한 GRASP_TS 알고리즘을 제시하며, 제시된 알고리즘을 널리 알려 진 benchmark graph 들에 적용함으로써 알고리즘의 효용성을 보이고자 한다.

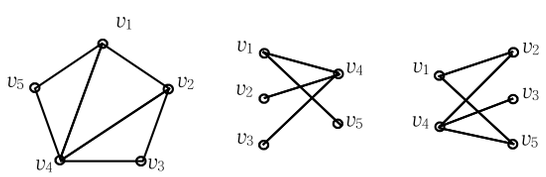
본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2 장에서는 본 연구에서 제시되는 GRASP_TS 알고리즘에 대한 자세한 내용을 제시하며, 3 장에서는 제시된 알고리즘을 다양한 그래프들에 대하여 적용한 실험 결과 및 분석을 수행한다. 끝으로 4 장에서는 결론을 맺는다.

2. MBSP를 위한 GRASP + Tabu Search (GRASP_TS) 알고리즘

X를 주어진 최적화 문제인 P의 해집합 (*solution space*)이라 하고 $x \in X$라 하자. 또한 P를 최대값 계산 문제 (*Maximization Problem*) 라 하고 x^*를 P의 최적해 (*optimum solution*) 그리고 $f(x)$를 P 의 목적함수 (*fitness function*)라 하자. $N(x) \subseteq X$는 x의 *neighborhood* 라 불리우며 $x' \in N(x)$는 x에 특정한 연산과정을 거쳐 생성되는 해로서 x의 *neighbor* 라 불리운다. Tabu Search (TS)는 지역탐색 알고리즘 (*Local Search Algorithm*) 과 유사하게 해집합으로부터 한 개 해인 x를 선택한 후 $N(x)$내 해들 중 최적의 neighbor 인 x'로 이동 (move) 한 후 이러한 과정을 반복적으로 수행함으로써 점진적으로 x^*에 도달하는 메타 휴리스틱 알고리즘 (Metaheuristic Algorithm) 이다. TS의 특징은 지역탐색 알고리즘과는 달리 $N(x)$를 구성할 때 최근에 고려되었던 해들은 제외시킨다는 것이다. 이를 위하여 TS는 *tabu-list* 라는 고정된 크기의 메모리를 이용하여 최근에 고려되었던 해들을 tabu-list 에 저장한 후 $N(x)$를 구성할 때 tabu-list 에 등록되어 있는 해들을 제외시킨다. 이를 통하여 TS는 유사한 해들이 순환 (cycling) 되는 것을 방지하며 따라서, 지역최적해 (*local optimum solution*) 로부터 벗어날 수 있는 메카니즘을 제공하게 된다.

Greedy Randomized Adaptive Search Procedure (GRASP)는 초기해인 x를 생성한 후 x에 대하여 지역탐색 알고리즘을 적용하는 과정을 반복하는 메카니즘으로 구성되어 있다. TS는 일반적으로 초기해를 무작위로 생성하는 반면에 GRASP 는 초기해를 탐욕 알고리즘 (*Greedy Algorithm*)을 이용하여 생성함으로서 초기해의 목적함수 값을 향상시키는 특징 및 다양한 초기해를 고려함으로써 지역최적해로부터 벗어날 수 있는 특징을 갖고 있다.

Fig. 2의 GRASP_TS는 본 연구에서 제안하고 있는 MBSP를 위한 알고리즘으로서, *iterationGRASP* 개의 초기해에 대하여 TS 및 Path-relinking 을 반복적으로 적용하는 메카니즘으로 구성되어 있다. 주어진 그래프를 $G = (V, E)$라 하고 $V = \{v_1, v_2, ..., v_n\}$ 이라 하자. GRASP_TS는 V 를 X 및 Y ($X \neq \emptyset$, $Y \neq \emptyset$) 로 분할하면서 cutsize의 크기가 최대인 bipartition을 계산하는 것을 목적으로 하고 있다. $s = (v_1v_2\cdots v_n)$는 길이 n 의 0-1 vector 로서 한 개 해인 bipartition을 나타낸다. 즉, $v_i = 0$ 이면 정점 v_i는 X로 분할된 것을 나타내며, $v_i = 1$ 이면 정점 v_i는 Y 로 분할된 것이다. $s[v_i]$ 는 s 로부터 정점 v_i의 값이라 하자. $f(s)$는 s 로부터 생성되는 bipartition 의 cutsize 크기를 나타내며, f^* 및 s^* 는 각각 최적해의 cutsize 의 크기 및 bipartition 을 나타낸다. 예를 들어, Fig. 1(a)의 5 개 정점으로 구성된 그래프로부터 $s_1 = (0, 0, 0, 1, 1)$ 및 $s_2 = (0, 1, 1, 0, 1)$ 을 두 개 해라 하자. 그렇다면 s_1 은 G 를 $X = \{v_1, v_2, v_3\}$ 및 $Y = \{v_4, v_5\}$ 로 분할한 것이며, s_2 는 $X = \{v_1, v_4\}$, $Y = \{v_2, v_3, v_5\}$ 로 분할한 것이다. Fig. 1의 (b) 및 (c) 는 s_1 및 s_2 로부터 G 가 이분 그래프로 분할된 상태를 보여주며 이로부터 $f(s_1) = 4$ 및 $f(s_2) = 5$ 이므로 s_2 가 s_1 보다 우월한 해임을 알 수 있다. G 에 대하여 $f^* = 5$ 이며 따라서 s_2 는 G 의 최적해이지만 일반적으로 최적해는 유일하지 않다. 예를 들어, $s_3 = (0, 1, 0, 1, 1)$ 이라하면 $f(s_3) = 5$ 이므로 s_3 ($\neq s_2$) 또한 최적해이다.



(a) G (b) s_1 (c) s_2

Fig. 1. 그래프 $G = (V, E)$ 및 두 개 해 $s_1 = (0, 0, 0, 1, 1)$, $s_2 = (0, 1, 1, 0, 1)$. $f(s_1) = 4$ 및 $f(s_2) = 5$.

```
procedure GRASP_TS( iterationGRASP, iterationTS )
1  f* = -∞
2  for ( i = 0; i < iterationGRASP; i++ )
3        s = generate_Greedy_Solution( );
4        s' = TabuSearch( s, iterationTS );
5        s'' = Path-Relinking( s' );
6        update_EliteSet( s'' );
7        if ( f( s'' ) > f* )
8              f* = f( s'' );
9              s* = s'';
10 endfor
11 return s*
end procedure
```

Fig. 2. GRASP_TS

$v \in V$에 대하여 $X(v) = \{u \in V | uv \in E, u \in X\}$, $Y(v) = \{u \in V | uv \in E, u \in Y\}$ 및 $nX(v) = |X(v)|$, $nY(v) = |Y(v)|$ 라 하고, 양의 정수 m에 대하여 $Random(m)$은 0~(m - 1) 범위 내의 정수 난수를 생성하는 함수라 하자. generate_Greedy_Solution() 은 다음과 같은 과정을 통하여MBSP 를 위한 초기해를 생성한다. 알고리즘의 시작 단계에서 모든 $v \in V$ 에 대하여 $X(v)$ 및 $Y(v)$는 Ø 으로 초기화되며 (따라서, $nX(v) = nY(v) = 0$), $v \in V$ 에 대하여 $nX(v) > nY(v)$ 라 하자. 그렇다면 x를 Y 로 분할함으로써 x를 X로 분할하는 경우와 비교하여 cutsize 의 크기를 $nX(v) - nY(v)$ (> 0) 개 만큼 증가시킬 수 있게 된다. 만일 $nX(v) = nY(v)$ 라면 v 를 임의로 X 또는 Y로 분할한다. Fig. 3은 generate_Greedy_Solution() 의 pseudo-code 를 보여준다.

```
procedure generate_Greedy_Solution( )
1  V' = V; X(v) = Y(v) = Ø for all v ∈ V.
2  while ( V' ≠ Ø )
3        v ∈ V';
4        if ( nX(v) > nY(v) )
5              s[v] = 1;
6        else if ( nX(v) < nY(v) )
7              s[v] = 0;
8        else
9              s[v] = Random(2);
10       V' = V' \ {v};
11 endwhile
12 return s;
end procedure
```

Fig. 3. generate_Greedy_Solution

Fig. 4의 TabuSearch()는 generate_Greedy_Solution() 에 의하여 생성된 초기해인 s에 대하여 tabu search 를 적용하는 알고리즘으로서$N(s)$로부터 특정 조건을 만족하는 새로운 해인 s' 를 선택한 후 s' 가 s를 대체하게 된다. s의 neighbor 는 다음과 같이 정의된다. 해 $s = (v_1 v_2 \cdots v_n)$ 에 대하여 s의 neighbor 인 $s_i = (v_1, v_2, ..., \overline{v_i}, ...v_n)$ 는 s로부터 한 개 정점 v_i ($1 \le i \le n$)를 다른 쪽 partite set 으로 이동시킨 것으로 정의하며, $N(s) = \{s_1, s_2, ..., s_n\}$ 는 s의 neighborhood 로서 s의 n 개 neighbor 들의 집합이다. TabuSearch() 는 $N(s)$로부터 가장 큰 크기의 cutsize 를 갖는 neighbor 를 선택하지만 정점들이 너무 빈번히 X 및 Y 사이를 이동하는 것을 금지시키며 이는 TabuSearch() 가 순환적으로 유사한 해들 사이에서 이동하는 것을 방지하려는 목적을 가지고 있다. 이를 위하여 TabuSearch() 는TL 이라는 vector 를 이용한다. 즉, TL 은 tabu-list 의 역할을 수행하며 $TL[v] = t$ 라면 정점 v 는 TabuSearch() 의 t 번째 반복연산에서 다른 쪽 partite set 으로 이동하였던 것을 나타낸다. 따라서, $s_i \in N(s)$ 에 대하여 Fig. 4의 (5) 에서 $T - TL[v_i] < tabuTenure$ 라면 v_i 는 $tabuTenure$ 기간 내에 자신이 포함되었던 원래 partite set 으로 다시 복원하는 것이므로 이러한 s_i 는 고려대상으로부터 제외된다. 그러나 (6) 에서 만일 $f(s_i) > f^*$ 라면, 즉, s_i 의 cutsize 크기가 지금까지 발견된 가장 우월한 해인 f^* 의 cutsize 의 크기보다 크다면 비록 s_i 가 tabu-list 에 포함되었다 하여도 s_i를 고려대상에 포함시키며 이러한 메카니즘은 *aspiration criteria* 라고 알려져 있다. Fig. 4의 pseudo-code 에 포함되지는 않았지만 만일 (3) 의 for loop 에서 s_i가 선택되지 못하는 경우가 발생하면 (16) 에서 s 는 임의의 s_i ($1 \le i \le n$) 로 대체된다.

c를 $N(s)$ 로부터 선택된 해라 하자. TabuSearch() 에서 $f(c) > f(s)$ 라는 보장은 없으며 simulated annealing 과 유사하게 본 조건을 이용하여 TabuSearch() 가 tabu-list 와 함께 지역 최적해로부터 벗어날 수 있는 메카니즘을 확보하게 된다.

(6) 및 (10) 에서 $f(s_i)$ 를 계산하는 과정은 다음과 같다. s 로부터 v_i ($1 \le i \le n$) 에 대하여 $id(v_i) = |\{u \in V | uv_i \in E, s[u] = s[v_i]\}|$ 및 $od(v_i) = |\{u \in V | uv_i \in E, s[u] \ne s[v_i]\}|$ 라 하자. 예를 들어, $v \in X$ 라 하면 $id(v)$ 는 s 에서 정점 v 가 X 내에 인접한 정점개수를 나타내며, $od(v)$ 는 v 가 Y 내에 인접한 정점개수를 나타낸다. 정점 v 에 대하여 $id(v) > od(v)$ 라 하자. 그러한 경우 v 를 다른 쪽 partite set 으로 이동시키면 cutsize 는 $\Delta = id(v) - od(v)$ (> 0) 개 만큼 증가하게 된다. 따라서, $f(s_i) = f(s) + (id(v) - od(v))$ 이다. (3) ~ (14) 사이의 계산복잡도는 다음과 같다. $\Delta = id(v) - od(v)$ 라 하고 $T(\Delta)$ 를 Δ 를 계산하는 과정에 요구되는 계산량이라 하고 $d(v)$ 를 G 에서 정점 v 의 차수 (*degree*) 라 하자. $id(v)$ 및 $od(v)$ 는 v 에 인접한 모든 정점들을 정확히 1 번씩만 고려함으로써 계산이 가능하므로 $T(\Delta) \in d(v) \in n$ 이다. 그러나, $|N(s)| = n$ 이므로 (3) ~ (14) 사이의 foreach loop 의 계산량은 $O(n^2)$ 을 요구한다.

```
procedure TabuSearch( s, iterationTS )
1  c* = -∞
2  for ( T = 0; T < iterationTS; T++ )
3        for each ( s_i ∈ N( s ) )// s_i = (v_1, v_2, ..., v_i, ...v_n)
4            if ( id( v_i ) - od( v_i ) > 0 )
5                  if ( T - tabuTenure ≤ TL[ i ] )
6                        if ( f( s_i ) > f* )
7                              c = s_i ;
8                              c* = f(c);
9                              tabuNode = i ;
10                 elseif ( f( s_i ) > c* )
11                       c = s_i ;
12                       c* = f(c)
13                       tabuNode = i ;
14       endforeach
15       TL[ tabuNode ] = T ;
16       s = c;
17 endfor
18 return s
end procedure
```

Fig. 4. TabuSearch

EliteSet 을 GRASP_TS 의 실행 도중에 발견된 우월한 해들의 집합이라 하고, *nEliteSet* = |*EliteSet*| 및 $x \in$ EliteSet 이라 하자. Fig. 5의 Path-relinking 은 Fig. 2의 (4)에서 tabu search 의 결과로 생성된 해 s를 x 와 동일한 해로 변환시키는 과정에서 더욱 우월한 해를 생성하고자 하는 메카니즘이다. 따라서, x 는 s 에 대한 안내자 (*guiding solution*) 의 역할을 수행한다. 이를 위하여 Fig. 5의 (5)~(8) 을 통하여 s 및 x 로부터 서로 다른 partite set 에 포함되어 있는 정점들을 추출하여 이를 Z에 저장시킨다. $Z = \{i_1, i_2, ..., i_m\}$ 이라 하면 s로부터 m 개 neighbor 인 s_{i_1}, s_{i_2}, ..., s_{i_m} 등이 정의된다. 따라서, choose_Best(Z) 는 다음과 같은 정점을 선택한다.

$$\text{choose\_Best}(Z) = \arg\max\{f(s_{i_1}), f(s_{i_2}), ..., f(s_{i_m})\}$$

예를 들어, n = 5 라 하고 s = (0, 0, 1, 0, 1) 및 x = (1, 0, 0, 1, 1) 이라 하자. 그렇다면 Fig. 5의 (5) ~ (8) 로부터 Z = {1, 3, 4} 이다. 만일 $f(s_1)$ = 1, $f(s_3)$ = 2, $f(s_4)$ = 3 이라면 choose_Best(Z) = 4 이다. Fig. 5의 (10)~(17) 사이의 while loop 는 Z 내 모든 정점들을 다른 쪽 partite set 으로 차례대로 이동시키면서 가장 우월한 해를 c 에 저장한다.

```
procedure Path-Relinking( s )
1   x ∈ EliteSet;
2   Z = Ø;
3   c* = f(s);
4   c = s;
5   for ( i = 0; i < n; i++ )
6       if ( s_i ≠ x_i )
7           Z = Z ∩ { i };
8   endfor
9   Δ = |Z|;
10  while ( Δ > 1 )
11      j = choose_Best( Z );
12      if ( f( s_j ) > c* )
13          c = s_j;
14          c* = f( s_j );
15      Z = Z \ { j };
16      Δ = Δ - 1;
17  endwhile
18  return c ;
endprocedure
```

Fig. 5. Path-Relinking

Fig. 2.의 update_EliteSet(s') 은 *EliteSet*을 관리하는 기능을 수행하며 본 연구에서는 [9] 에서 제시된 관리방법과 유사한 방법을 다음과 같이 적용한다. *Eliteset* 에는 모두 *nEliteSet* 개의 우월한 해를 저장하며 최초에 *EliteSet* = Ø 이므로 tabu search 를 통하여 생성된 처음 *nEliteSet* 개 해들은 단순히 *EliteSet* 에 저장된다. *EliteSet* 에 *nEliteSet* 개의 해가 저장되어 있다 하고 x를 Path-Relinking 을 통하여 생성된 해라 하자. 또한 c_B 및 c_W를 각각 EliteSet 내의 최선 및 최악의 해라 하자. 만일 $f(x) > f(c_B)$ 라면 x 는 c_W 를 대체한다. $f(x) < f(c_B)$ 및 $f(x) > f(c_W)$ 인 경우에는 x 가 EliteSet 내에 중복된 해가 없다면 x 는 c_W 를 대체한다.

3. 실험 및 분석

$G = (V, E)$ 를 그래프라 하고 $|E| = m$ 이라 하자. 또한 *MBSP*(G) 를 G 에 대한 최적해가 포함하는 cutsize 의 크기라 하고 *GRASP_TS*(G) 를 G 에 대하여 GRASP_TS 가 최종적으로 도출하는 cutsize 의 크기라 하자. MBSP 를 위한 benchmark graph 들은 아직 알려진 것이 없으므로 본 연구에서는 GRASP_TS 의 효율성을 검증하기 위하여 두 가지 종류의 그래프들을 선택하였다. 첫 번째는 이분그래프로서 G를 이분그래프라 하면 MBSP(G) = m 이 성립한다. 따라서, 이분그래프들에 대해서는 GRASP_TS(G) 및 m 의 차이를 통하여 GRASP_TS 의 효율성을 검증하는 것이 가능하다. 두 번째는 일반그래프(general graph) 로서 다른 그래프 문제들을 위하여 DIMACS 에서 개발된 benchmark graph 들을 대상으로 하였다. 본 실험에 사용된 그래프들은 인터넷을 통하여 쉽게 검색할 수 있으며 [10, 11], GRASP_TS 는 Java (Version 1.7.0_21) 언어로 구현하였다. 실험 장비는 4ZB 의 메모리 및 2.69GHz 의 성능을 갖춘 데스크 탑 컴퓨터를 이용하였다.

3.1 이분그래프에 대한 실험 및 분석

본 절에서는 GRASP_TS 가 최적해에 얼마만큼 접근할 수 있는지를 검사하기 위하여 GRASP_TS 를 이분그래프에 적용한 실험결과를 포함하고 있다. $G = (V, E)$가 이분그래프인 경우 명백히 MBSP(G) = $|E|$ 이므로 GRASP_TS 의 효율성은 MBSP(G) 및 GRASP_TS(G) 의 차이로서 검증될 수 있다.

Table 1의 10개 그래프들은 *bip* 계열의 그래프로서 본 그래프들은 *Steiner problem* 을 위하여 [12] 에서 제시한 benchmark graph 들의 일부로서 이들은 모두 이분그래프이다. 본 연구에서는 이들 10개 그래프들에 대하여 *iterationGRASP* = 10, *iterationTS* = 1,000, *tabuTenure* = 20 및 *maxEliteSet* = 10 의 파라메타를 이용하여 GRASP_TS 를 각 그래프에 대하여 100번씩 적용한 결과 GRASP_TS 는 100% 의 비율로 bip 계열의 그래프들에 대하여 최적해를 계산한다는 것을 발견하였다. GRASP_TS 는 bip 계열의 그래프들에 대하여 항상 최적해를 계산하기 때문에 GRASP_TS 가 계산한 cutsize 의 크기를 제시하는 것은 무의미하므로 본 실험에서는 초기해로부터 얼마나 빨리 최적해를 계산하는지를 측정하였다.

Table 1에서 '*n*' 및 '*m*' 은 각각 *G* 의 정점 및 간선 개수를 나타내며, '초기해' 는 generate_Greedy_Solution() 이 생성한 초기해들의 평균 cutsize 의 크기를 나타낸다. 본 그래프들에 대하여 MBSP(*G*) = *m* 이며, '초기해/*m*' 은 초기해/*m* 으로서 초기해가 최적해에 접근한 비율을 % 로 나타낸다. '*iGRASP*' 및 '*iTS*' 는 각각 GRASP_TS 가 최적해를 계산한 시점에서의 *iterationGRASP* 및 *iterationTS* 값을 나타낸다. 끝으로 '*Time*' 은 평균 계산시간을 초단위로 나타낸 것이다. Table 1로부터 GRASP_TS 는 최적해와 비교하여 평균적으로 70% 대의 초기해로부터 2 회 이내의 *iterationGRASP* 를 통하여 최적해를 계산할 수 있음을 알 수 있다. 또한 계산에 요구되는 시간이 매우 작은 것은 bip 계열의 그래프들은 이분그래프이므로 이들의 간선밀도(*edge density*) 가 낮기 때문이라고 판단된다.

Table 1. *bip* 계열의 그래프들에 대한 GRASP__TS 의 실험결과
tabuTenure = 20, *iterationGRASP* = 10, *iterationTS* = 1000, *maxEliteSet* = 10

그래프	*n*	*m*	초기해	초기해/*m*(%)	*iGRASP*	*iTS*	*Time*
bip42p	1,200	3,982	2,850	72	1.4	441	0.01
bip42u	1,200	3,982	2,837	71	1.2	476	0.01
bip52p	2,200	7,997	5,711	71	1.1	731	0.03
bip52u	2,200	7,997	5,750	72	1.4	736	0.02
bip62p	1,200	10,002	7,434	74	1	277	0.01
bip62u	1,200	10,002	7,609	76	1	271	0.01
bipa2p	3,300	18,073	13,942	77	1.3	729	0.04
bipa2u	3,300	18,073	13,797	76	1.6	743	0.07
bipe2p	550	5,013	4,597	92	1	57	0.01
bipe2u	550	5,013	4,125	82	1	98	0.01

3.2 DIMACS 의 benchmark 그래프들에 대한 실험 및 분석

일반그래프에 대한 실험은 DIMACS 에 제시된 그래프들을 두 개 그룹으로 나누어 실험하였다. 첫번째 그룹은 *DSJC* 계열의 그래프로서 *graph coloring* 문제를 위한 benchmark graph 로서 가장 널리 사용되고 있는 그래프들이다. 실험은 각 그래프들에 대하여 GRASP_TS 를 10 번씩 적용하였으며 파라메타는 *tabuTenure* = 20, *nEliteSet* = 10, *iterationGRASP* = 40 및 *iterationTS* = 5,000 을 적용하였다. Table 2는 실험결과를 보여주며 '*n*' 및 '*m*' 은 각각 그래프의 정점 및 간선 개수를 나타내며, '*Best*', '*Worst*' 및 '*Ave*' 는 각각 10 회의 실험 중 도출된 최선, 최악 및 평균적인 cutsize 의 크기를 나타낸다. '*Time*' 은 1 회의 실험에 소요된 시간의 평균값을 초로 나타낸 것이다. Table 2의 '*TSOnly*' 는 *iterationGRASP* = 1, *iterationTS* = 200,000 을 적용하고 Path-Relinking() 을 제거한 상태에서 GRASP_TS 를 각 실험 그래프에 대하여 10 회의 실험을 통하여 도출된 평균적인 cutsize 의 크기를 나타낸다. 따라서, '*TSOnly*' 는 Tabu Search 과정만을 적용한 것으로서 본 결과를 통하여 Tabu search 만을 적용하는 것보다는 GRASP+TS 가 더욱 우월한 해를 계산할 수 있다는 것을 입증하려는 목적을 가지고 있다.

Table 2. DSJC 계열의 그래프들에 대한 GRASP__TS 의 실험결과
tabuTenure = 20, *nEliteSet* = 10, *iterationGRASP* = 40, *iterationTS* = 5000.

그래프	*n*	*m*	*Best*	*Worst*	*Ave*	*Time*	*TSOnly*
DSJC125.1	125	736	517	517	517	0.16	517
DSJC125.5	125	3,891	2,211	2,211	2,211	0.32	2,211
DSJC125.9	125	6,961	3,659	3,659	3,659	0.45	3,659
DSJC250.1	250	3,218	2,054	2,054	2,054	0.31	2,053
DSJC250.5	250	15,668	8,608	8,608	8,608	0.61	8,607
DSJC250.9	250	27,897	14,446	14,446	14,446	0.9	14,446
DSJC500.1	500	12,458	7,491	7,489	7,490	0.61	7,482
DSJC500.5	500	62,624	33,460	33,460	33,460	1.29	33,447
DSJC500.9	500	112,437	57,588	57,588	57,588	1.94	57,580
DSJC1000.1	1,000	49,629	28,383	28,340	28,363	1.32	28,321
DSJC1000.5	1,000	249,826	130,999	130,922	130,952	2.93	130,899
DSJC1000.9	1,000	449,449	228,518	228,479	228,494	4.43	228,474

Table 2로부터 DSJC125.*n*, DSJC250.*n* 및 DSJC500.9 에 대하여 *Best* = *Ave* 임을 알 수 있다. 따라서, 동일한 파라메타를 이용하여 이들 7 개 그래프들에 대한 별도의 실험에서 각 그래프에 대하여 100 회씩의 실험을 반복하였지만 결과는 동일하였다. 이로부터 이들 그래프에 대하여 GRASP_TS는 최적해에 가까운 결과를 도출한 것으로 판단된다. 또한 GRASP_TS 의 결과를 '*TSOnly*' 의 결과와 비교하면 모든 그래프들에 대하여 GRASP+TS 의 결과가 tabu search 만을 이용한 것보다 우월함을 알 수 있다. 본 결과로부터 GRASP+TS 는 tabu search 와 비교하여 복수개의 초기해로부터 새로운 탐색을 시작하기 때문에 지역 최적해로부터 벗어날 수 있는 기회를 제공하는 메카니즘이라고 판단된다. 특히 Table 3 은 그래프의 크기가 증가할 수록 tabu search 와 비교하여 GRASP+TS 가 더욱 우월한 해를 계산할 수 있다는 것을 보여준다.

Table 3. DIMACS 계열의 그래프들에 대한 GRASP__TS 의 실험결과
tabuTenure = 20, *nEliteSet* = 10, *iterationGRASP* = 100, *iterationTS* = 10000.

그래프	*n*	*m*	*Best*	*Worst*	*Ave*	*Time*	*TSOnly*
r1000.1	1,000	14,378	8,535	8,519	8,526	4.76	8,472
r1000.1c	1,000	485,090	249,558	249,558	249,558	20.63	249,305
c1000.9	1,000	450,079	228,833	228,826	228,831	19.4	228,800
c2000.5	2,000	999,836	517,143	517,001	517,092	29.81	516,783
c2000.9	2,000	1,799,532	910,268	910,172	910,209	44.69	910,126
c4000.5	4,000	4,000,268	2,047,717	2,047,351	2,047,477	77.42	2,047,212
keller6	3,361	4,619,898	2,470,352	2,470,352	247,0352	72.63	2,464,120
MANN_a81	3,321	5,506,380	2,756,227	2,756,171	275,6191	88.23	2,756,015
frb100-40	4,000	572,774	327,004	327,003	327,003	30.46	326,439

두번째 실험 그룹은 DIMACS 계열의 그래프들 중 정점 개수가 1,000 ~ 4,000 개 범위 내에 해당하는 그래프들로서 본 그래프들에 대하여 *tabuTenure* = 20, *nEliteSet* = 10, *iterationGRASP* = 100 및 *iterationTS* = 10,000 을 이용하여 실험하였으며 Table 3은 해당 실험 결과를 보여 준다. Table 3.의 *'TSOnly'*는 Table 2와 유사하게 *iterationGRASP* = 1, *iterationTS* = 1,000,000 및 Path-Relinking() 을 제거한 상태에서 실험한 결과값이다. 본 실험을 통하여 GRASP_TS 는 실행속도가 높은 알고리즘임을 알 수 있다. 예를 들어, Table 3의 MANN_a81 은 간선밀도가 99.9% 인 그래프이지만 실행속도는 88.23 초를 요구하므로 실행속도가 매우 높다는 것을 확인할 수 있다.

4. 결 론

본 연구에서는 MBSP를 위한 Tabu Search 및 GRASP를 조합한 메타휴리스틱 알고리즘을 제시하였으며, 제시된 알고리즘을 이분그래프 및 일반그래프에 대하여 실험함으로서 알고리즘의 효율성을 검증하였다. 그러나 *Variable Neighborhood Search*(*VNS*) 알고리즘과의 조합 또한 가능하다고 판단되므로 후속 연구로서 VNS 를 이용하여 초기해를 생성함으로써 제시된 알고리즘의 효율을 더욱 높이고자 한다. 또한 이론적인 연구를 통하여 MBSP 의 하한 및 상한값을 제시할 수 있는 노력이 필요하다고 판단된다.

Reference

[1] R. M. Karp, "*Reducibility among combinatorial problems*," in Complexity of Computer Computations. New York: Plenum, pp.85-104. 1972.

[2] S. Even and Y. Shiloach, "*NP-completeness of several arrangement problems*," Dept. Computer Science, Technion, Haifa, Israel, Tech. Rep. 43, 1975.

[3] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "*Some simplified NP-complete graph problems*," Theoretical Computer Science, Vol.1, pp.237-267, 1976.

[4] J. A. Bondy and S. C. Locke, "*Largest bipartite subgraph in triangle-free graphs with maximum degree three*," J. Graph Theory, Vol.10, pp.477-504, 1986.

[5] Kuo Chun Lee, Nobuo Funabiki, and Yoshiyasu Takefuji, "*A parallel improvement algorithm for the bipartite subgraph problem*", IEEE Transactions on Neural Networks, Vol.3, No.1, pp.139-145, 1992.

[6] Rong Long Wang, Zheng Tang, and Qi Ping Cao, "*A hopfield network learning method for bipartite subgraph problem*", IEEE Transactions on Neural Networks, Vol.15, No.6, 2004.

[7] Fred Glover, "*Future paths for integer programming and links to artificial intelligence*", Computers and Operations Research 13(5), pp.533-549, 1986.

[8] T.A. Feo and M.G.C. Resende, "*A probabilistic heuristic for a computationally difficult set covering problem*", Operations Research Letters, 8, pp.67-71, 1989.

[9] P. Festa, P. M. Pardalos, M. G. C. Resende, and C. C. Ribeiro, "*Randomized heuristics for the max-cut problem*," Optimization Methods and Software, 7, pp.1033-1058, 2002.

[10] DIMACS: http://dimacs.rutgers.edu/Challenges/

[11] SteinLab: http://steinlib.zib.de/steinlib.php

[12] I. Rosseti, M. P. de Aragão, C. C. Ribeiro, E. Uchoa, and R. F. Werneck, *New benchmark instances for the steiner problem in graphs*, In Extended Abstracts of the 4th Metaheuristics International Conference, pp.557-561, Proto, Portugal, 2001.

한 근 희

e-mail : kehan@kongju.ac.kr

1986년 건국대학교 물리학과(학사)

1992년 Univ. of Central Oklahoma 응용수학과(이학석사)

1996년 Univ. of Oklahoma 컴퓨터과학과 졸업(이학박사)

1996년~2000년 한국전자통신연구원

1999년~2000년 미국 NIST 초빙연구원

2000년~현 재 공주대학교 응용수학과 교수

관심분야 : 그래프 알고리즘, Heuristic Algorithm

김 찬 수

e-mail : chanskim@kongju.ac.kr

1991년 부산대학교 전산통계학과(학사)

1997년 부산대학교 통계학과(이학박사)

2002년~현 재 공주대학교 응용수학과 교수

관심분야 : 베이지안 네트워크, Heuristic Algorithm