Journal of Heuristics, 8: 449–465, 2002

Bounds and Heuristics for the Shortest Capacitated Paths Problem

MARIE-CHRISTINE COSTA
CEDRIC, CNAM, 292 rue Saint-Martin, 75003 Paris, France

ALAIN HERTZ AND MICHEL MITTAZ
Department of Mathematics, EPFL, 1015 Lausanne, Switzerland

Abstract

Given a graph G, the Shortest Capacitated Paths Problem (SCPP) consists of determining a set of paths of least total length, linking given pairs of vertices in G, and satisfying capacity constraints on the arcs of G.

We formulate the SCPP as a 0-1 linear program and study two Lagrangian relaxations for getting lower bounds on the optimal value. We then propose two heuristic methods. The first one is based on a greedy approach, while the second one is an adaptation of the tabu search meta-heuristic.

Key Words: minimum cost integer multicommodity flow problem, bandwidth packing problem, Lagrangian relaxation, tabu search

1. Introduction

Let $G = (V, E)$ be an undirected graph where $V = \{v_1, \ldots, v_n\}$ is the vertex set and E the edge set. A positive length c_{ij} is associated to each edge (v_i, v_j) in E. Consider a set $C = \{(s_1, t_1), \ldots, (s_K, t_K)\}$ containing K pairs of vertices in G. The *edge-disjoint paths problem* consists of determining whether there exist K mutually edge-disjoint paths in G linking the pairs of vertices in C. This problem is known to be NP-complete (Middendorf and Pfeiffer, 1993; Vygen, 1995).

The *shortest disjoint paths problem* consists of finding K edge-disjoint paths of least total length, linking all pairs of vertices in C. This problem is at least as difficult as the edge-disjoint paths problem described above and is therefore NP-hard.

Assume now that a positive weight σ_k is associated to each pair (s_k, t_k) in C. By extension, we will say that a path linking s_k to t_k has weight σ_k. In addition, assume that each edge $(v_i, v_j) \in E$ has a capacity u_{ij}. *The shortest capacitated paths problem* (*SCPP*) consists of determining K paths of least total length linking all pairs of vertices in C, and such that the total weight of the paths going through any edge $(v_i, v_j) \in E$ does not exceed its capacity u_{ij}. The SCPP is NP-hard since it includes the shortest disjoint paths problem as a special case, which is obtained by setting $u_{ij} = 1$ for all edges (v_i, v_j) in E and $\sigma_k = 1$ for all pairs (s_k, t_k) in C.

Applications of the SCPP arise naturally in several contexts, e.g. in VLSI-design. Our study has been motivated by a real-life problem at EDF (Electricité de France) dealing with the optimisation of the layout of cables in a power plant. Solutions methods actually used by EDF determine cable paths one after the other, without any global vision of the data. A reduction in the total cable length may help saving large sums of public money.

The SCPP can be viewed as a special case of the minimum cost integer multicommodity flow problem. Ahuja, Magnanti, and Orlin (1993) have published a very complete survey on the continuous multicommodity flow problem which is now well solved (McBride and Mamer, 1997). A fast approximation algorithm has recently been proposed by Klein et al. (1994) for an integer multicommodity flow problem with unit capacities. However, there are apparently very few results concerning the problem including costs (or lengths) and integrality constraints, even with demands (or weights) equal to one. Specific algorithms have been proposed for particular cases, such as ring networks (Vachani et al., 1996). Notice that if all pairs in C are equal, and all weights are equal to one, then the problem becomes a minimum cost flow problem, and is therefore polynomially solvable.

This paper is organised as follows. In Section 2, we formulate the SCPP as an 0-1 linear program. We then study two possible Lagrangian relaxations and compare the different lower bounds obtained by solving the Lagrangian dual problems. Since the SCPP is NP-hard, we focus in Section 3 on heuristic methods. We first generate various orderings of the pairs in C; for each such ordering, we use a greedy approach that builds the paths one after the other. As second heuristic approach, we propose a tabu search algorithm. Computational results are presented in Section 4. We show in Section 5 how a simple variation of the SCPP can be used as a basic tool for the solution of the bandwidth packing problem in telecommunication networks (Cox, Davis, and Qiu, 1991).

2. A 0-1 linear program and Lagrangian relaxations

In this section we first formulate the SCPP as a 0-1 linear program. For each edge (v_i, v_j) in E and each pair (s_k, t_k) in C, we define a Boolean variable x_{ijk} which is equal to one if (v_i, v_j) belongs to the path linking s_k to t_k, and zero otherwise. Since G is undirected, we have $x_{ijk} = x_{jik}$ and we can therefore only consider variables x_{ijk} with $i < j$.

For each vertex v_i in V and each pair (s_k, t_k) in C, we consider the Boolean variable y_{ik} which is equal to one if and only if vertex v_i belongs to the path linking s_k to t_k. Hence, y_{ik} is necessarily equal to one if v_i is equal to s_k or t_k. The SCPP can then be formulated as follows.

$$\text{Minimise} \quad \sum_{k=1}^{K} \sum_{\substack{(v_i,v_j)\in E \\ i<j}} c_{ij} x_{ijk}$$

$$\text{Subject to} \quad \sum_{k=1}^{K} \sigma_k x_{ijk} \le u_{ij} \quad \forall (v_i, v_j) \in E, i < j \tag{1}$$

$$\sum_{\substack{(v_i,v_j)\in E \\ j<i}} x_{jik} + \sum_{\substack{(v_i,v_j)\in E \\ i<j}} x_{ijk} = 2y_{ik} \quad \forall k \in \{1, \ldots, K\}, \forall v_i \neq s_k, t_k \tag{2}$$

$$\sum_{\substack{(v_i,v_j)\in E\\ j<i}} x_{jik} + \sum_{\substack{(v_i,v_j)\in E\\ i<j}} x_{ijk} = 1 \quad \forall k \in \{1,\ldots,K\}, v_i = s_k \text{ or } t_k \tag{3}$$

$$x_{ijk} = 0 \text{ or } 1 \quad \forall k \in \{1,\ldots,K\}, \forall (v_i, v_j) \in E, i < j \tag{4}$$

$$y_{ik} = 0 \text{ or } 1 \quad \forall k \in \{1,\ldots,K\}, \forall v_i \in V \tag{5}$$

Constraints (1) ensure that the capacity constraints are not violated while constraints (2) and (3) impose that each pair (s_k, t_k) in C is linked by a path.

We now study two Lagrangian relaxations of this 0-1 linear program. We first relax constraints (1), and then constraints (2) and (3). Readers not familiar with Lagrangian relaxations are referred to Fisher (1981).

2.1. *Capacity constraints relaxation*

Let us associate a non-negative Lagrangian multiplier λ_{ij} to each edge $(v_i, v_j) \in E, i < j$. The Lagrangian relaxation of constraints (1) induces the following Lagrangian subproblem.

$$\text{Minimise} \quad \sum_{k=1}^{K} \sum_{\substack{(v_i,v_j)\in E\\ i<j}} c_{ij} x_{ijk} - \sum_{\substack{(v_i,v_j)\in E\\ i<j}} \lambda_{ij} \left(u_{ij} - \sum_{k=1}^{K} \sigma_k x_{ijk} \right)$$

Subject to constraints (2), (3), (4) and (5).

The optimal value of this problem will be denoted $L_1(\lambda)$. The function to be minimised can equivalently be written as follows.

$$\sum_{k=1}^{K} \sum_{\substack{(v_i,v_j)\in E\\ i<j}} (c_{ij} + \lambda_{ij}\sigma_k) x_{ijk} - \sum_{\substack{(v_i,v_j)\in E\\ i<j}} \lambda_{ij} u_{ij}$$

The second term of this function does not depend on variables x_{ijk} and y_{ik}. We can therefore ignore it during the minimisation process. Moreover, the above relaxed problem has no bundle constraints, linking variables associated with different values of k. Hence, the Lagrangian subproblem can be decomposed into K independent subproblems. The subproblem associated with the k-th element of C is defined as follows.

$$\text{Minimise} \quad \sum_{\substack{(v_i,v_j)\in E\\ i<j}} (c_{ij} + \lambda_{ij}\sigma_k) x_{ijk}$$

$$\text{Subject to} \quad \sum_{\substack{(v_i,v_j)\in E\\ j<i}} x_{jik} + \sum_{\substack{(v_i,v_j)\in E\\ i<j}} x_{ijk} = 2y_{ik} \quad \forall v_i \neq s_k, t_k$$

$$\sum_{\substack{(v_i,v_j)\in E\\ j<i}} x_{jik} + \sum_{\substack{(v_i,v_j)\in E\\ i<j}} x_{ijk} = 1 \quad \text{for } v_i = s_k \text{ or } t_k$$

$$x_{ijk} = 0 \text{ or } 1 \quad \forall (v_i, v_j) \in E, i < j$$

$$y_{ik} = 0 \text{ or } 1 \quad \forall v_i \in V$$

This is a shortest path problem in an undirected graph. Since the lengths, weights and Lagrangian multipliers are non-negative (i.e., $c_{ij} + \lambda_{ij}\sigma_k \geq 0$), each subproblem can be solved by means of Dijkstra's algorithm (see for example Section 4.5 in Ahuja, Magnanti, and Orlin (1993)).

In summary, $L_1(\lambda)$ can be determined by solving K independent shortest path problems. The optimal value Z_1 of the following Lagrangian multiplier problem is a lower bound on the optimal value of the SCPP:

$$Z_1 = \max_{\lambda \geq 0} L_1(\lambda)$$

This lower bound can be obtained by means of classical subgradient optimisation methods or multiplier ascent methods.

2.2. *Paths constraints relaxation*

Instead of relaxing constraints (1), we study here the Lagrangian multiplier problem obtained by relaxing the paths constraints, that is constraints (2) and (3). We associate Lagrangian multipliers μ_{ik} to each vertex v_i in V and each pair (s_k, t_k) in C. The relaxed problem to be solved has the following form.

$$\begin{aligned}
\text{Minimise} \quad & \sum_{k=1}^{K} \sum_{\substack{(v_i,v_j)\in E \\ i<j}} c_{ij} x_{ijk} \\
& + \sum_{k=1}^{K} \sum_{\substack{v_i \in V \\ v_i \neq s_k, t_k}} \mu_{ik} \left(2y_{ik} - \sum_{\substack{(v_i,v_j)\in E \\ j<i}} x_{jik} - \sum_{\substack{(v_i,v_j)\in E \\ i<j}} x_{ijk} \right) \\
& + \sum_{k=1}^{K} \sum_{\substack{v_i \in V \\ v_i \in \{s_k, t_k\}}} \mu_{ik} \left(1 - \sum_{\substack{(v_i,v_j)\in E \\ j<i}} x_{jik} - \sum_{\substack{(v_i,v_j)\in E \\ i<j}} x_{ijk} \right) \\
\text{Subject to} \quad & \text{constraints (1), (4) and (5).}
\end{aligned}$$

The optimal value of this Lagrangian subproblem will be denoted $L_2(\mu)$. The above function can be rewritten in the following much simpler form.

$$\sum_{k=1}^{K} \sum_{\substack{(v_i,v_j)\in E \\ i<j}} (c_{ij} - \mu_{ik} - \mu_{jk}) x_{ijk} + 2 \sum_{k=1}^{K} \sum_{\substack{v_i \in V \\ v_i \neq s_k, t_k}} \mu_{ik} y_{ik} + \sum_{k=1}^{K} \sum_{\substack{v_i \in V \\ v_i \in \{s_k, t_k\}}} \mu_{ik}$$

The last term of this function can be ignored during the minimisation process. Moreover, since constraints (1) do not depend on variables y_{ik}, we can set y_{ik} equal to one if the corresponding Lagrangian multiplier μ_{ik} is negative, and zero otherwise. Notice that the above relaxed problem does not contain bundle constraints linking variables associated with different edges of E. The Lagrangian subproblem can therefore be decomposed into $|E|$ independent subproblems. The subproblem associated with edge (v_i, v_j) of E is defined as follows.

$$\begin{aligned} \text{Minimise} \quad & \sum_{k=1}^{K}(c_{ij} - \mu_{ik} - \mu_{jk})x_{ijk} \\ \text{Subject to} \quad & \sum_{k=1}^{K}\sigma_k x_{ijk} \leq u_{ij} \\ & x_{ijk} = 0 \text{ or } 1 \quad \forall k \in \{1, \ldots, K\} \end{aligned}$$

This is a knapsack problem which is known to be NP-hard, but for which efficient exact solution methods have been developed (Martello and Toth, 1990). Notice that if all weights σ_k are equal to one, then an optimal solution of the knapsack problem can easily be determined. Indeed, let I be the subset of $\{1, \ldots, K\}$ containing all indices k such that $c_{ij} - \mu_{ik} - \mu_{jk}$ is negative. If I contains at most u_{ij} elements, then set $x_{ijk} = 1$ if and only if $k \in I$. Otherwise, sort I according to non-decreasing values of $c_{ij} - \mu_{ik} - \mu_{jk}$, and set $x_{ijk} = 1$ if and only if k is among the u_{ij} first elements in I.

The optimal value Z_2 of the following Lagrangian multiplier problem is also a lower bound on the optimal value of the SCPP:

$$Z_2 = \max_{\mu \in \mathbb{R}} L_2(\mu)$$

This lower bound can also be obtained by means of classical subgradient optimisation techniques.

2.3. *Comparison of bounds*

Let Z_c denote the optimal value of the linear programming problem obtained by relaxing the integrality constraints of the SCPP. It is well-known that a lower bound obtained by a Lagrangian relaxation technique is at least as sharp as Z_c (see for example Section 16.4 in Ahuja, Magnanti, and Orlin (1993)). Hence, we have $Z_c \leq Z_1$ and $Z_c \leq Z_2$.

A Lagrangian bound may be equal to a linear programming bound. Such a situation occurs if the Lagrangian subproblem satisfies a property, known as the *integrality property* (see for example Section 16.4 in Ahuja, Magnanti, and Orlin (1993)). A Lagrangian subproblem satisfies the integrality property if, given any choice of coefficients in the objective function, it has an integer optimal solution even if the integrality constraints are relaxed.

The Lagrangian subproblems defined at Sections 2.1 and 2.2 do not satisfy the integrality property. Indeed, consider first the Lagrangian subproblem obtained by relaxing constraints (1). We have seen that this problem can be solved by means of K independent shortest path problem. The following example shows that the optimal value of a shortest path problem can be strictly larger than the optimal value of its continuous relaxation.

Consider a graph G with vertex set $\{a, b, c, d\}$ and edge set $\{(a, b), (b, c), (c, d)\}$. Assume that each edge has length 1. The problem consisting in finding the shortest path from a to d in G can be formulated as follows.

$$\begin{aligned} \text{Minimise} \quad & x_{ab} + x_{bc} + x_{cd} \\ \text{Subject to} \quad & x_{ab} + x_{bc} = 2y_b \\ & x_{bc} + x_{cd} = 2y_c \\ & x_{ab} = 1, x_{cd} = 1 \\ & x_{bc}, y_b, y_c = 0 \text{ or } 1 \end{aligned}$$

The optimal value of this problem is 3 while the continuous relaxation has a minimum value of 2, obtained by setting $x_{bc} = 0$ and $y_b = y_c = 0.5$.

We have shown in Section 2.2 that the Lagrangian subproblem obtained by relaxing constraints (2) and (3) can be solved by means of $|E|$ independent knapsack problems. Again, the optimal value of a knapsack problem can be strictly larger than the optimal value of its continuous relaxation. Indeed, consider the following example.

$$\begin{aligned} \text{Minimise} \quad & -x_1 - x_2 \\ \text{Subject to} \quad & 2x_1 + 2x_2 \le 3 \\ & x_1, x_2 = 0 \text{ or } 1 \end{aligned}$$

In this example, the optimal value is -1 while the continuous relaxation has a minimum value of -1.5.

The situation is different if all weights σ_k in the SCPP are equal to 1. Indeed, we have seen in Section 2.2 that in such a case, $L_2(\mu)$ has an optimal integer solution, even if we relax the integrality constraints. We have therefore $Z_1 \ge Z_c = Z_2$, which means that the first lower bound Z_1 is always as least as sharp as the second one Z_2.

3. Heuristic methods

We describe in this section two heuristic methods for the solution of the SCPP. The first one has been proposed by Turki (1997), and is based on a greedy approach, while the second one is a tabu search algorithm. Both heuristics use the same following basic concepts. A solution of the SCPP is defined as a set of paths $P_k (1 \le k \le K)$ linking each pair (s_k, t_k) of vertices in C. Notice that we do not impose that a solution satisfies the capacity constraints. A solution is called *feasible* if the total weight of the paths going through any edge $(v_i, v_j) \in E$ does not exceed its capacity u_{ij}. A *partial* solution S (possibly infeasible) is a set of paths linking only a subset of pairs of vertices in C. We denote $I(S)$ the set of indices k such that s_k is linked to t_k by a path P_k in S.

Let S be a partial feasible solution, and let (s_k, t_k) be a pair of vertices in C which is not linked by a path in S (i.e., $k \notin I(S)$). We define the following graph $G(S, (s_k, t_k))$ which indicates how a path linking s_k to t_k can be added to S without violating the capacity constraints. The graph $G(S, (s_k, t_k))$ is obtained from G by removing all edges (v_i, v_j)

such that, given the routes in S, the residual capacity on (v_i, v_j) is not large enough for an additional route linking s_k to t_k and going through (v_i, v_j). More precisely, the edge (v_i, v_j) is removed from G if $\sigma_k + \sum_{\substack{r \in I(S)\\(v_i,v_j)\in P_r}} \sigma_r > u_{ij}$.

The proposed heuristic methods work with two different objective functions. The first, $F_1(S)$ is simply the total length of solution S. As we allow infeasible solutions during the search process, we also define an artificial objective $F_2(S) = F_1(S) + \alpha\Delta(S)$, where α is a self adjusting parameter (see Section 3.2) and $\Delta(S)$ is the total overload on the edges, that is:

$$\Delta(S) = \sum_{(v_i,v_j)\in E} \text{Max}\left\{0, \left(\sum_{\substack{k\in I(S)\\(v_i,v_j)\in P_k}} \sigma_k - u_{ij}\right)\right\}$$

We denote F_1* and F_2* the best known values of F_1 and F_2, respectively.

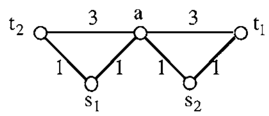
3.1. *A multi-start greedy approach*

The first heuristic described in this paper has been developed by Turki (1997), and uses a greedy approach which, given an ordering of the pairs of vertices in C, builds the paths one after the other, taking care of not violating the capacity constraints. If the greedy procedure can successfully determine K paths, then the solution provided by this algorithm is necessarily feasible. Various randomly generated orderings are given as input to the greedy procedure. This multi-start greedy heuristic is called GREEDY(N), where N is the number of different orderings submitted to the greedy procedure. It is described in figure 1.

Notice that the above algorithm does not necessarily produce a feasible solution, even if such a solution exists and all permutations of the pairs in C are tested. Indeed, consider

```
Procedure GREEDY(N)
  1. Set the iteration counter c:=0, S:=∅ and F1*:=infinity.
  2. If c = N then STOP.
     Else Permute the pairs of vertices in C and set the path counter k:=1.
  3. Let (sk,tk) be the k-th pair in C.
     Determine the shortest path Pk linking sk to tk in G(S,(sk,tk)).
     If such a path exists then
            add Pk to S.
            If k=K and F1(S)<F1* then set F1*:=F1(S) and S*:=S.
     Else set c:=c+1, S:=∅ and go to 2.
  4. If k=K then set c:=c+1, S:=∅ and go to 2.
     Else set k:=k+1 and go to 3.
```

Figure 1. A greedy algorithm for the SCPP.



C={(s1,t1),(s2,t2)}.
The edge lengths are indicated on the edges.
All weigths and capacities are equal to one.

Figure 2. A bad example for GREEDY(N).

the example in figure 2. The unique feasible solution is made of paths $P_1 = \{(s_1, a), (a, t_1)\}$ and $P_2 = \{(s_2, a), (a, t_2)\}$. The GREEDY($n$) algorithm does not find such a solution since the shortest path from s_1 to t_1 goes through s_2, and the shortest path from s_2 to t_2 goes through s_1.

3.2. *A tabu search approach*

We now describe an adaptation of the tabu search technique to the SCPP. Readers not familiar with this meta-heurisitic are referred to Reeves (1993). The proposed algorithm handles solutions that are not necessarily feasible. Each violation of the capacity constraints is penalised. The use of large penalties helps intensifying the search in feasible regions of the search space, while small penalties tend to diversify the search towards new regions of the search space.

The tabu search algorithm, called TABU_SCPP uses two improvement procedures. The first one, called 1-OPT tries to improve a feasible solution S by replacing paths in S by shorter ones. We take care of not violating the capacity constraints. More precisely, let (s_k, t_k) be a pair of vertices in C such that the path P_k linking s_k to t_k in S is not the shortest possible one. We first construct a partial solution S' obtained from S by removing P_k. We then determine the shortest path linking s_k to t_k in $G(S', (s_k, t_k))$. If this path is shorter than P_k, we add it to S' to build the new solution S. This process is repeated until no path in S can be improved. The pseudo-code of this improvement procedure is given in figure 3.

```
Procedure 1-OPT(S,S*)
  1. Set k:=1, F1*:=F1(S) and S*:=S
  2. Let Pk be the path linking sk to tk in S.
     Construct the partial solution S' obtained from S by removing Pk.
     Determine the shortest path P linking sk to tk in G(S',(sk,tk)).
     If P is shorter than Pk then add P to S' and set S:=S'.
  3. If k=K then
            If F1(S)<F1* then go to 1.
            Else STOP.
     Else set k:=k+1 and go to 2.
```

Figure 3. First improvement procedure.

Procedure 2-OPT(S,S*)

1. Set r:=1, q:=2, F_1*:=F_1(S) and S*:=S
2. Let P_r and P_q be the paths linking s_r to t_r and s_q to t_q in S, respectively.

   Construct the partial solution S' obtained from S by removing P_r and P_q.
   Determine the shortest path P linking s_r to t_r in G(S',(s_r,t_r)) and add P to S'.
   Determine the shortest path P' linking s_q to t_q in G(S',(s_q,t_q)) and add P' to S'.
   If F_1(S')<F_1(S) then set S:=S'.

   Remove P and P' from S'.
   Determine the shortest path linking s_q to t_q in G(S',(s_q,t_q)) and add it to S'.
   Determine the shortest path linking s_r to t_r in G(S',(s_r,t_r)) and add it to S'.
   If F_1(S')<F_1(S) then set S:=S'.
3. **If** q=K **then**
   If r<K-1 **then** set r:=r+1, q:=r+1 and go to 2.
   Else STOP.
   Else set q:=q+1 and go to 2.

Figure 4. Second improvement procedure.

The second improvement algorithm, called 2-OPT, removes two paths from the current solution S and tries to replace them by paths of smaller total length. Here again, we take care of not violating the capacity constraints. When two paths P_r and P_q, linking s_r to t_r and s_q to t_q, are removed, we determine two shortest possible paths by first connecting s_r to t_r before s_q to t_q, and then by connecting s_q to t_q before s_r to t_r. This algorithm is described in figure 4.

The above two improvement procedures are included in TABU_SCPP which is now described in details. Let S be a solution, (s_k, t_k) a pair of vertices in C, and (v_i, v_j) an edge on the path P_k linking s_k to t_k in S. Let S' be the partial solution obtained from S by removing P_k, and let α be the penalty parameter used in the definition of the objective function F_2. We define a new graph $H(S, (s_k, t_k), (v_i, v_j))$ which is obtained from G as follows. We first remove the edge (v_i, v_j) from G. To each other edge (v_a, v_b) in G, we add α to c_{ab} if, given the routes in S', the residual capacity on (v_a, v_b) is not large enough for a route linking s_k to t_k and going through (v_a, v_b). More precisely, the new graph has the same vertex set V as G, while its edge set is equal to $E\backslash\{(v_i, v_j)\}$. The length c'_{ab} of an edge in $H(S, (s_k, t_k), (v_i, v_j))$ is defined as follows:

$$c'_{ab} = \begin{cases} c_{ab} + \alpha & \text{if } \sigma_k + \sum\limits_{\substack{r \in I(S') \\ (v_a, v_b) \in P_r}} \sigma_r > u_{ij} \\ c_{ab} & \text{otherwise} \end{cases}$$

The solution obtained by adding to S' the shortest path P linking s_k to t_k in $H(S, (s_k, t_k), (v_i, v_j))$ is called a *neighbour solution* of S. Notice that P is necessarily different from the path P_k in S since it does not contain edge (v_i, v_j). Moreover, the new path P linking s_k to t_k tries to avoid going through edges having not enough residual capacity.

A neighbour solution S' of S is thus obtained by modifying a path linking a pair (s_k, t_k) of vertices of C. When moving from S to S', the pair (s_k, t_k) is declared tabu for θ iterations, where θ is randomly selected in a given interval $[\theta_1, \theta_2]$. By extension, a solution obtained by modifying the path linking a tabu pair of vertices is called a *tabu solution*. This variable tabu list length strategy was inspired from Taillard's work (1991). After extensive experiments on the application of tabu search to the quadratic assignment problem, this author concludes that the probability of obtaining a global optimum is increased in the case of a variable list length. According to preliminary experiments, we use $\theta_1 = \lceil \frac{K}{6} \rceil$ and $\theta_2 = \lceil \frac{K}{3} \rceil$ in our implementation.

The neighbourhood $N(S)$ of S can be very large and we have therefore decided to explore only part of it. This is simply done by randomly generating a fixed number M of neighbours and choosing the best one which is not tabu. A low value of M tends to produce low quality solutions; in contrast, running times become excessive with high values of M. As a compromise, we use $M = \lceil \frac{K}{5} \rceil$ in our implementation.

As mentioned at the beginning of Section 3, the artificial objectif function F_2 depends on a penalty parameter α. All too often, choosing an appropriate value of α is difficult, and a wrong choice can have an adverse impact on the performance of the algorithm. Therefore, as suggested by Hertz (1992), we define α as a self adjusting parameter. Initially, α is set equal to 1. Every MODIF_α iterations, α is halved if all previous MODIF_α solutions were feasible and doubled if they were all infeasible. We found the algorithm is not very sensitive to the value of MODIF_α. We use Modif_$\alpha = 20$ in our implementation.

The search process ends when the number of consecutive iterations without improvement of F_1* or F_2* reaches a given value Max_Iter. If Max_Iter is too low, some good solutions will be missed. If it is too high, there is a risk that the algorithm will run for a long time without improvement. Sensitivity analysis performed on test problems suggest that Max_Iter = 500 is a good compromise.

The pseudo-code of the TABU_SCPP algorithm is given in figure 5.

4. Computational results

The GREEDY and TABU_SCPP algorithms were coded in C and run on a Silicon Graphics Indigo2 machine (195 MHz, IP28 processor). They were tested on 18 different problem types which are summarised in Table 1. For each problem type, ten instances were generated according to a procedure described in Turki (1997). This gives a total of 180 instances which range in size from $n = 100$ to 1000 vertices, and from $K = 25$ to 1000 pairs of vertices in C. The procedure in Turki (1997) which generates instances can be described as follows. The vertices are first randomly generated in the $[0, 1]^2$ square. A first set of edges is generated by constructing a random spanning tree on these vertices. Additional edges are then randomly generated until the total number of edges is equal to $\frac{3n}{2}$. Hence, the average number of neighbours of each vertex is equal to three. The weights σ_k of the pairs (s_k, t_k) of vertices in C are either set equal to one unit (in problem types 1–10), or randomly generated according to a discrete uniform distribution on [1, 10] (in problem types 11–18).

Algorithm TABU_SCPP

(Initial Solution)

1. Determine the shortest path P_k ($1 \le k \le K$) linking s_k to t_k in G and set $S=\{P_1,...,P_K\}$.
   If $\Delta(S)=0$ **then** STOP : S is an optimal solution.
   Else, set the iteration counter c:=0, F_1*:=infinity, F_2*:=F_2(S) and S_2*:=S.

(Next Iteration)

2. Set c:=c+1.

Generate M neighbours $S_1',...,S_M'$ of S, where each S_h' ($1 \le h \le M$) is obtained from S as follows : choose a pair (s_k,t_k) in C and an edge (v_i,v_j) on the path P_k linking s_k to t_k in S; Build S_h' from S by first removing P_k, and then adding the shortest path linking s_k to t_k in $H(S,(s_k,t_k),(v_i,v_j))$.

A tabu neighbour S_h' is discarded unless it is feasible and $F_1(S_h')<F_1^*$, or it is infeasible and $F_2(S_h')<F_2^*$. Let N'(S) be the subset of remaining neighbours.

Determine the best neighbour S' in N'(S) according to function F_2.

If S' is feasible while S was not, **then** go to 3.
If S' is feasible and $F_1(S')<F_1^*$, **then** go to 3.

Let (s_k,t_k) be the pair of vertices in C such that S' is obtained from S by modifying the path linking s_k to t_k. Choose a random number θ in $[\theta_1,\theta_2]$, declare (s_k,t_k) tabu for θ iterations and go to 4.

(Incumbent Improvement)

3. call 1-OPT(S',S") and then call 2-OPT(S",S').

(Increment Updates)

4. **If** S' is feasible **and** $F_1(S')<F_1^*$ **then** set S_1^*:=S', $F_1^*=F_1(S')$ and Best_Iter:=c.
   If $F_2(S')<F_2^*$ **then** set S_2^*:=S', $F_2^*=F_2(S')$ and Best_Iter:=c;
   If (c mod Modif_α) = 0 **then** update α.
   Set S:=S'.
   If (c - Best_iter) = Max_Iter **then** go to 5. Else go to 2.

(Final Improvement)

5. Set S:=S_1* and call 2-OPT(S,S_1*)
   If $F_1(S_1^*)<F_1^*$ **then** set $F_1^*=F_1(S_1^*)$ and repeat 5.
   Else STOP.

Figure 5. A tabu search algorithm for the SCPP.

Table 1. Description of the problem types.

Problem type	Number n of vertices	Number K of pairs in C	Weights σ_k	Capacities u_{ij}
1	100	25	1	Random choice in [1, 5]
2	100	50	1	Random choice in [1, 10]
3	100	75	1	Random choice in [1, 15]
4	100	100	1	Random choice in [1, 25]
5	200	50	1	Random choice in [1, 5]
6	200	100	1	Random choice in [1, 15]
7	200	150	1	Random choice in [1, 25]
8	200	200	1	Random choice in [1, 35]
9	500	500	1	Random choice in [1, 50]
10	1000	1000	1	Random choice in [1, 50]
11	100	100	Random choice in [1, 10]	Random choice in [1, 200]
12	200	200	Random choice in [1, 10]	Random choice in [1, 300]
13	500	500	Random choice in [1, 10]	Random choice in [1, 350]
14	1000	1000	Random choice in [1, 10]	Random choice in [1, 70]
15	100	100	Random choice in [1, 10]	Random choice in [1, U]
16	200	200	Random choice in [1, 10]	Random choice in [1, U]
17	500	500	Random choice in [1, 10]	Random choice in [1, U]
18	1000	1000	Random choice in [1, 10]	Random choice in [1, U]

If the edge capacities are not large enough, then the SCPP has no feasible solution. Given an instance of the SCPP, let U be the smallest integer such that GREEDY(100) is able to determine a feasible solution, assuming that the capacities u_{ij} of the edges (v_i, v_j) are randomly selected according to a discrete uniform distribution in $[1, U]$. Two kinds of edge capacities were considered. In a first set of experiments, the capacities were randomly chosen in the interval $[1, U]$ (problem types 15–18). A second set of easier instances were generated by randomly selecting each edge capacity in the interval $[1, x]$, where x is strictly larger than U (problem types 1–14).

In Table 2, we compare the results produced by TABU_SCPP with those obtained by the multi-start greedy algorithm GREEDY(N), by setting $N = 500$ and $N = 5000$. The 2-OPT procedure used in TABU_SCPP induces very large CPU-times for large size problems. We have therefore implemented a simplified version of TABU_SCPP, in which no call to 2-OPT is performed in Steps 3 and 5. This modified version of the tabu search algorithm is called FAST_TABU.

The results are summarised in Table 2. For each problem type, we report the average and worst percentage deviations of the heuristic solution value over the lower bound Z_1 defined in Section 2.1. The CPU-times are given in seconds.

It clearly appears that optimal or near-optimal solutions can easily be obtained for instances having unit weights σ_k (i.e., problem types 1–10). Indeed the GREEDY algorithm produces solutions which are on average less than 0.5% away from the lower bound, and

Table 2. Computational results.

Problem type	GREEDY(500)			FAST_TABU			GREEDY(5000)			TABU_SCPP		
	Average deviation	Worst deviation	CPU time	Average deviation	Worst deviation	CPU time	Average deviation	Worst deviation	CPU time	Average deviation	Worst deviation	CPU time
1	0.007	0.061	5	0.007	0.061	3	0.007	0.061	60	0.007	0.061	6
2	0.400	1.470	12	0.200	1.149	7	0.104	0.354	121	0.001	0.003	13
3	0.615	1.894	17	0.083	0.314	10	0.273	0.972	178	0.014	0.140	29
4	0.281	0.800	24	0.084	0.298	14	0.102	0.365	243	0.001	0.002	30
5	0.154	0.739	24	0.012	0.093	19	0.031	0.303	246	0.001	0.002	33
6	0.282	0.850	49	0.027	0.109	27	0.152	0.556	500	0.003	0.031	71
7	0.452	0.957	76	0.056	0.139	47	0.318	0.634	751	0.002	0.018	147
8	0.699	2.081	99	0.137	0.467	62	0.586	1.779	983	0.058	0.406	217
9	0.535	0.977	692	0.093	0.223	415	0.452	0.935	6945	0.003	0.013	2126
10	0.434	0.693	3416	0.081	0.171	2099	0.379	0.685	35215	0.006	0.016	13526
11	1.079	2.319	23	1.131	1.975	12	0.944	1.922	236	0.887	1.661	27
12	0.647	1.236	100	0.550	1.698	64	0.545	1.005	1010	0.426	1.021	149
13	0.874	1.259	693	0.527	0.819	569	0.775	1.251	6959	0.427	0.708	1584
14	4.706	5.460	3273	2.940	3.918	6281	4.559	5.366	33282	2.254	3.111	36393
15	2.045	4.048	23	1.771	3.173	19	1.685	2.781	233	1.455	2.480	39
16	1.374	2.604	95	0.915	1.700	69	1.200	2.098	965	0.780	1.371	216
17	1.799	3.454	679	1.064	1.790	602	1.652	3.145	6830	0.828	1.146	2692
18	13.249	21.695	2735	8.123	12.723	10366	13.026	21.023	26955	6.587	10.950	64983

this deviation is never worse than 1.7%. The situation is even better with TABU_SCPP since this algorithm is able to determine solutions with an average deviation not larger than 0.05%, and a worst deviation of 0.4%.

The heuristic solution values of the problems in which the edge weights are randomly generated in the interval [1, 10] (i.e. problem types 11–18) have a larger deviation over the lower bound Z_1. Relatively larger gaps can be observed for instances where the edge capacities are randomly selected according to a discrete uniform distribution in $[1, U]$ (i.e., problem types 15–18). However, the TABU_SCPP algorithm has systematically smaller gaps than the GREEDY algorithm. Notice also that part of this gap may be due to the difference between the lower bound and the global optimal value.

While TABU_SCPP is faster then GREEDY(5000) for instances having up to 500 vertices, the situation is the opposite for larger size instances. This phenomenon can easily be explained by the use of the 2-OPT improvement algorithm in TABU_SCPP. Indeed, the 2-OPT procedure solves $O(K^2)$ shortest path problems on graphs with n vertices. These shortest paths are determined by means of the $O(n^2)$ Dijkstra's algorithm. Since K is proportional to n, it follows that the 2-OPT procedure has an $O(n^4)$ complexity. For comparison, GREEDY(N) has to solve N shortest path problems, which gives an $O(Nn^2)$ complexity.

The FAST_TABU algorithm avoids the use of the 2-OPT procedure, and its CPU-time is therefore competitive when compared with GREEDY(5000), even for large size instances. Notice also that FAST_TABU generally provides better solutions than GREEDY(5000). However, while FAST_TABU is faster than TABU_SCPP, it gives solution values with larger deviations over the lower bound Z_1.

Since the CPU-time of GREEDY(N) increases linearly with N, one can easily get a very fast heuristic method for the SCPP by choosing small values for N. It can however be noticed that while the GREEDY(500) algorithm has about the same CPU-times as FAST_TABU, it produces solutions with noticeably larger average deviations.

5. Relation with the bandwidth packing problem

The *bandwidth packing problem* (BWP) arises in the area of telecommunications, and has first been introduced by Cox, Davis, and Qiu (1991). It can be described as follows. Consider a set $C = \{(s_1, t_1), \ldots, (s_K, t_K)\}$ containing K pairs of vertices in a capacitated network G. Each pair (s_k, t_k) $(1 \leq k \leq K)$ corresponds to a call request, with source node s_k, terminal node t_k, non-negative bandwidth requirement σ_k, and known revenue r_k. Each call (s_k, t_k) can be assigned only to a node-simple path P in G, and such an assignment induces a *profit*

$$\pi_k = r_k - \sigma_k \sum_{(v_i, v_j) \in P} c_{ij}$$

where c_{ij} is a unit cost on edge (v_i, v_j) in G.

The BWP consists of selecting a subset of calls in C, and of assigning the selected calls to paths in the network, while satisfying capacity restrictions on G and maximising the total profit. More precisely, consider the Boolean variable z_k which is equal to one if and only if call (s_k, t_k) is selected in C in order to be routed in G. The BWP can then be formulated as follows:

$$\text{Maximise} \quad \sum_{k=1}^{K} r_k z_k - \sum_{k=1}^{K} \sigma_k \sum_{\substack{(v_i,v_j)\in E \\ i<j}} c_{ij} x_{ijk}$$

$$\text{Subject to} \quad \text{constraints (1), (2), (4) and (5) of Section 2}$$

$$\sum_{\substack{(v_i,v_j)\in E \\ j<i}} x_{jik} + \sum_{\substack{(v_i,v_j)\in E \\ i<j}} x_{ijk} = z_k \quad \forall k \in \{1, \ldots, K\},\ v_i = s_k \text{ or } t_k \tag{3'}$$

$$z_k = 0 \text{ or } 1 \quad \forall k \in \{1, \ldots, K\} \tag{6}$$

Proposed solution methods for the BWP are based on tabu search (Anderson et al., 1993; Laguna and Glover, 1993), genetic algorithms (Cox, Davis, and Qiu, 1991), column generation (Parker and Ryan, 1995) and integer programming (Park, Kang, and Park, 1996).

Given a subset of selected calls in C, the *bandwidth packing subproblem* (BWPS) consists of finding the best routing of these selected calls. More precisely, assume without loss of generality that $z_k = 1$ for $k = 1, \ldots, K'$, and $z_k = 0$ for $k = K' + 1, \ldots, K$. The BWPS can then be formulated as follows:

$$\text{Maximise} \quad \sum_{k=1}^{K'} r_k - \sum_{k=1}^{K'} \sigma_k \sum_{\substack{(v_i,v_j)\in E \\ i<j}} c_{ij} x_{ijk}$$

Subject to constraints (1), (2), (3), (4) and (5) of Section 2 (where K must be replaced by K')

The first term of the above objective function is constant and can therefore be ignored. Hence, the BWPS can be rewritten as the following constrained minimisation problem:

$$\text{Minimise} \quad \sum_{k=1}^{K'} \sum_{\substack{(v_i,v_j)\in E \\ i<j}} \sigma_k c_{ij} x_{ijk}$$

Subject to constraints (1), (2), (3), (4) and (5) of Section 2 (where K must be replaced by K')

The above problem is very close to the SCPP. The unique difference appears in the objective function: if the edge (v_i, v_j) belongs to the path linking s_k to t_k, then its cost is c_{ij} in the SCPP, and $\sigma_k c_{ij}$ in the BWPS.

It is not difficult to adapt to the BWPS all developments presented in Sections 2 and 3. For example, as noticed in Section 2.1, the Lagrangian relaxation of constraints (1) induces a Lagrangian subproblem which can be decomposed into K independent shortest path problems. If the original problem is the BWPS, then the cost of an edge $(v_i, v_j) \in E$ in these subproblems is equal to $\sigma_k(c_{ij} + \lambda_{ij})$ (to be compared with $c_{ij} + \lambda_{ij}\sigma_k$ for the SCPP). Similarly, we have observed that the Lagrangian relaxation of the path constraints induces a Lagrangian subproblem which can be decomposed into $|E|$ independent knapsack problems. When dealing with the BWPS, the cost of the k-th object must be set equal to $\sigma_k c_{ij} - \mu_{ik} - \mu_{jk}$ (to be compared with $c_{ij} - \mu_{ik} - \mu_{jk}$ for the SCPP).

In summary, a possible approach for the solution of the BWP consists of iteratively generating subsets of calls in C. For each such subset, the best routing in G of the selected calls can be determined by solving the BWPS which is a variation of the SCPP.

6. Conclusions

We have determined two lower bounds and developed two heuristic methods for the Shortest Capacitated Paths Problem. This problem arises in several applications, e.g. in VLSI-design. The first proposed heuristic method, called GREEDY(N), is based on a greedy approach and provides good solutions within reasonable computing times. However, we have shown that instances can easily be generated such that this greedy algorithm has not the slightest chance to find a feasible solution to the SCPP, while such a solution exists.

We have then developed a tabu search algorithm, called TABU_SCPP, which provides solution values with a very small average deviation over a computed lower bound. The TABU_SCPP algorithm uses two improvement procedures, one of them being very time consuming for large size instances. We have therefore implemented a simplified version of TABU_SCPP, called FAST_TABU which does not use this improvement procedure. We have observed that FAST_TABU generally produces better solutions than GREEDY(5000), but is about ten times faster.

The above mentioned heuristic algorithms have been successfully used in a real-life context, for the layout of cables in a power plant. Moreover, as shown in Section 5, a simple variation of the SCPP can help for the solution of the more difficult BWP. Indeed, the BWP consists of selecting calls from a list of requests, and of routing these selected calls in a telecommunication network. For a given selection of calls, the best routing can be determined by solving the BWPS which is very close to the SCPP.

Acknowledgments

This Research was supported in part by the Swiss National Scientific Research Council, under grant 21-45574.95.

References

Ahuja, R., T. Magnanti, and J. Orlin. (1993). *Networks Flows, Theory, Algorithms, Applications.* Englewood Cliffs, NJ: Prentice Hall.

Anderson, C.A., K. Fraughnaugh, M. Parker, and J. Ryan. (1993). "Path Assignment for Call Routing: An Application of Tabu Search." *Annals of Operations Research* 41, 301–312.

Cox, L.A., L. Davis, and Y. Qiu. (1991). "Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks." In L. Davis (ed.), *Handbook of Genetic Algorithms*, New York: VanNostrand Reinhold.

Fisher, M.L. (1981). "The Lagrangian Relaxation Method for Solving Integer Programming Problems." *Management Science* 27, 1–18.

Gondran, M. and M. Minoux. (1985). *Graphes et Algorithmes*. Paris: Eyrolles.

Hertz, A. (1992). “Finding a Feasible Course Schedule Using Tabu Search.” *Discrete Applied Mathematics* 35, 255–270.

Kennington, J. (1978). “A Survey of Linear Cost Multicommodity Networks Flows.” *Opns. Res.* 26, 209–236.

Klein, P., S. Plotkin, C. Stein, and E. Tardos. (1994). “Faster Approximation Algorithms for the Unit Capacity Concurrent Flow Problem with Applications to Routing and Finding Sparse Cuts.” *SIAM J. on Comp.* 23(3), 466–487.

Korte, B., L. Lovasz, H.J. Prömel, and A. Schrijver. (1990). *Paths, Flows and VLSI-Layout.* Heidelberg: Springer-Verlag.

Laguna, M. and F. Glover. (1993). “Bandwidth Packing: A Tabu Search Approach.” *Management Science* 39(4), 492–500.

Martello, S. and P. Toth. (1990). *Knapsack Problems*. Chichester: Wiley.

McBride, R. and J. Mamer. (1997). “Solving Multicommodity Flow Problems with a Primal Embedded Network Simplex Algorithm.” *INFORMS J. on Comp.* 9(2), 154–162.

Middendorf, M. and F. Pfeiffer. (1993). “On the Complexity of the Disjoint Path Problem.” *Combinatorica* 13, 97–107.

Minoux, M. (1975). “Résolution des Problèmes de Multiflots en Nombres Entiers dans les Grands Réseaux.” *RAIRO Opns. Res.* 3, 21–40 (in French).

Park, K., S. Kang, and S. Park. (1996). “An Integer Programming Approach to the Bandwidth Packing Problem.” *Management Science* 42, 1277–1291.

Parker, M. and J. Ryan. (1995). “A Column Generation Algorithm for Bandwidth Packing.” *Telecommunications Systems* 2, 185–196.

Reeves, C. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Oxford: Blackwell.

Taillard, E. (1991). “Robust Taboo Search for the Quadratic Assignment Problem.” *Parallel Computing* 17, 433–445.

Turki, I. (1997). “Optimisation d’itinéraires de cables électriques,” Mémoire d’Ingénieur, CNAM-IIE, Paris.

Vachani R., A. Shulman, P. Kubat, and J. Ward. (1996). “Multicommodity Flows in Ring Networks.” *INFORMS J. on Comp.* 8(3), 235–242.

Vygen, J. (1995). NP-Completeness of Some Edge-Disjoint Paths Problem.” *Discrete Appl. Math.* 61, 83–90.