Multiple-Origin-Multiple-Destination Path Finding with Minimal Arc Usage: Complexity and Models

Roman Barták
Charles University in Prague
Prague, Czech Republic

Neng-Fa Zhou
CUNY Brooklyn College
New York, U.S.A.

Agostino Dovier
Università degli Studi di Udine
Udine, Italy

Abstract

The multiple-origin-multiple-destination (MOMD) problem is a simplified version of the logistics planning problem in which packages are required to be transported from their origins to their destinations by multiple trucks with a minimum total cost. This paper proves the NP-hardness of the problem, and gives two SAT-based models for solving the problem optimally. It also gives experimental results that compare these two SAT models and ASP and CP models.

Introduction

Given a weighted directed graph $G = (V, E)$, where V is a set of vertices, and E is a set of arcs each of which has an associated weight, and a set of origin-destination pairs, the *multiple-origin-multiple-destination (MOMD)* problem amounts to finding a subgraph G' of the minimum total weight that connects each origin-destination pair with a path. The resulting subgraph G' is guaranteed to be cycle-free if the weights are all non-negative. The Floyd-Warshall algorithm [2] for finding shortest paths is not applicable to the MOMD problem since it finds a shortest path for each of the pairs separately, and does not guarantee the minimality of the overall cost since it does not take into account shared arcs. Techniques for solving closely-related *cooperative path-finding (CPF)* problems [9] (also known as multi-agent path-finding problems) cannot be applied to MOMD due to a significance difference in problem constraints. While the CPF problem requires that two agents do not share the same link at the same time and usually the objective is minimizing makespan, the MOMD problem actually supports sharing the links as the objective is minimizing the total weight of used links.

The MOMD problem is a simplified version of the logics planning problem from the International Planning Competition 2014 [4], in which we can assume there are unlimited number of trucks each with unlimited capacity for transporting a set of packages from their origins to their destinations. The optimal solution to the MOMD problem provides a lower bound for the original logistics problem and hence it can be used for a better heuristic than the minimum-path heuristic used for the Transport problem.

The MOMD problem is also related to the road-building/maintenance problem, which designs/maintains a transportation network that satisfies connection requirements with the minimum cost. There exist techniques for special versions of the problem, for example, the multiple-origin-single-destination [10] (or the single-origin-multiple-destination, or the Steiner tree problem [3]), but we are unaware of any studies of the general MOMD problem.

In this paper, we will formally define the multiple-origin-multiple-destination problem and we will show that its decision variant is NP-complete. We will then propose an exact method to solve the problem by modeling it as a SAT problem. In particular, we will present two models of the problem, one based on flow constraints and the other based on reachability constraints. The optimization variant of the problem is then solved using the dichotomic branch-and-bound algorithm. The paper is concluded by experimental comparison of the models using the Picat system [8].

Problem Formulation and Complexity

The multiple-origin-multiple-destination (MOMD) path finding problem is formulated as follows. Assume a directed arc-weighted graph $G = (V, E, w)$, where V is a set of vertices, E is a set of directed arcs, and $w : E \rightarrow \mathbf{N}$ is a mapping of arcs to non-negative weights. Let P be a set of packages, such that each package $p \in P$ is defined as a pair $(orig(p), dest(p))$, where $orig(p) \in V$ is p's original location and $dest(p) \in V$ is p's destination location. The task is to select a subset $A \subseteq E$ of arcs such that for each package p there exists a directed path in the graph $G' = (V, A)$ from $orig(p)$ to $dest(p)$ and the sum of weights of arcs in A (that is $\sum_{a \in A} w(a)$) is minimal.

As mentioned in the Introduction, there exists a straightforward method to solve the MOMD problem non-optimally. First, find the shortest path for each package, for example using an all-pairs-shortest-path algorithm such as the Floyd-Warshall algorithm [2]. Let $sp(p)$ be the set of arcs used by the shortest path for package p. Then define A as $\cup_{p \in P} sp(p)$. Obviously, this method has a polynomial time complexity (the Floyd-Warshall algorithm has the time complexity $O(|V|^3)$ and there are even faster methods, for example, if the graph is sparse or the number of packages is small then the Dijkstra's algorithm is a better option). However, this method does no guarantee optimality as we will show in the section with experimental results. In this paper we focus on solving the MOMD problem optimally; we are not aware about any known method for the problem.

Let us first show that the MOMD problem is an NP-hard problem. We will assume the decision variant of the problem, where the task is for any given number k to verify that a set $A \subseteq E$ of arcs exists such that $\sum_{a \in A} w(a) \leq k$ and for each package p there exists a directed path in the graph $G' = (V, A)$ from $orig(p)$ to $dest(p)$. It is obvious that the decision variant of the MOMD problem belongs to the NP complexity class, because, given the set A, it is easy to verify in polynomial time that A is indeed a solution to the MOMD problem by finding a path for each package in the reduced graph $G' = (V, A)$.

The MOMD problem resembles the well known minimum Steiner tree problem [3], which is one of Karp's original 21 NP-complete problems. We will use the following general version of the minimum Steiner tree problem. We are given an edge-weighted graph $G = (V, E, w)$ and a subset $S \subseteq V$ of required vertices. A *Steiner tree* is a tree in G that spans all vertices of S. There are two variants of the problem: in the optimization problem, the task is to find a minimum-weight Steiner tree; in the decision problem, we are given a value k and the task is to decide if a Steiner tree of total weight at most k exists. We shall show now that the Steiner tree problem can be converted to the MOMD problem, which proves that the decision variant of the MOMD problem is an NP-complete problem.

Theorem 1. *The Steiner tree problem is reducible to the multiple-origin-multiple-destination problem in polynomial time.*

Proof. Let the Steiner tree problem be defined using the undirected edge-weighted graph $G = (V, E, w)$ and the subset $S \subseteq V$ of required vertices. For each non-directed edge $\{a, b\} \in E$ we introduce two directed arcs (a, b) and (b, a) both with the same weight as the original undirected edge, $\hat{w}((a, b)) = \hat{w}((b, a)) = w(\{a, b\})$. So we define the set $\hat{E}$ of arcs and mapping $\hat{w}$ to weights as follows:

$$\hat{E} = \{(a, b) \mid \{a, b\} \in E\}, \ \hat{w}((x, y)) = w(\{x, y\}).$$

Let $s \in S$ be any vertex. Then we define the set P of packages as follows:

$$P = \{(s, d) \mid d \in S \setminus \{s\}\}.$$

In other words, all packages are originated at some vertex s and their destinations are the other vertices from S (the case when S contains a single vertex is trivial). The MOMD problem then consists of the graph $\hat{G} = (V, \hat{E}, \hat{w})$ and the set P of packages. Notice that we actually define a single-origin-multiple-destinations problems, which is a special case of MOMD. Obviously this MOMD problem is generated in the polynomial time from the Steiner tree problem.

Now, we shall show that any Steiner tree of the cost at most k corresponds to a solution of the MOMD problem with the cost at most k and vice versa. Assume that T is a Steiner tree in G covering S. Then we can orient all edges in T in the direction away from s (start with the direct neighbors of s and continue away from s). This way we get a set $\hat{T}$ of directed arcs such that there is a directed path in $\hat{T}$ from s to each vertex $d \in S \setminus \{s\}$. Moreover, it holds $\sum_{a \in \hat{T}} \hat{w}(a) = \sum_{a \in T} w(a) = k$. Hence any Steiner tree T with cost k defines a solution $\hat{T}$ of the MOMD with cost k.

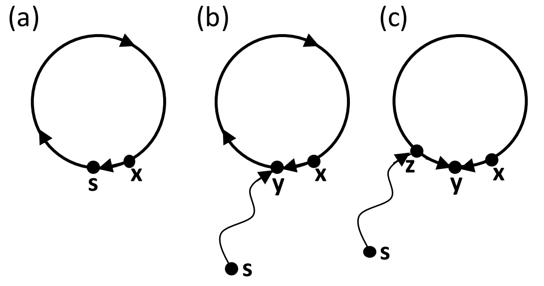

Figure 1: Cycles in the solution to a MOMD problem.

Let us assume now that we have a set $A \subseteq \hat{E}$ of arcs with the total cost k such that A is a solution to the above MOMD problem containing only the vertices reachable from s (arcs leading to vertices that are not reachable from s can be removed from A while still having a solution of the MOMD problem). Assume that A contains a directed cycle. If this cycle contains the vertex s (the vertex selected as the origin during the transformation) then there is also some arc $(x, s) \in A$ in the cycle (see Figure 1(a)). We can remove the arc (x, s) and the set $A \setminus \{(x, s)\}$ is a solution to the MOMD problem with a smaller cost (every vertex $y \in S \setminus \{s\}$ is still reachable from s). If the cycle does not contain the vertex s then the cycle contains a vertex y that is reachable from s without using any arc from the cycle (such vertex must exists as the cycle is reachable from s and s is not part of the cycle), see Figure 1(b). Let $(x, y) \in A$ be the arc from the cycle going to y. Then we can remove (x, y) from A and all vertices in A (including y) still remain reachable from s so the set $A \setminus \{(x, y)\}$ is a solution to the MOMD with a smaller cost. Assume now that A contains an undirected cycle (that is not a directed cycle). It means that there is some vertex y such that there are arcs $(x, y) \in A$ and $(z, y) \in A$ in the cycle. According to our assumption about A, both x and z are reachable from s. Without loss of generality let z be reachable from s by a path non-containing x. Then we can remove arc (x, y) and all vertices in S including y will still be reachable from s.

We just showed that if there exists a solution A with the cost k of the MOMD problem then it is possible to get a solution A' with the cost at most k such that this solution does not contain any cycle (directed or undirected) and all vertices used by arcs in A are reachable from s.

Now, let us define the set of arcs $T = \{\{a, b\} \mid (a, b) \in A'\}$. As all the vertices from arcs in A' are reachable and there is no cycle in A' then T must be a tree. Moreover $|T| = |A'|$ because there is no pair of arcs (x, y) and (y, x) in A (they would form a directed cycle). Hence $\sum_{a \in T} w(a) = \sum_{a \in A'} \hat{w}(a) \leq k$. Finally, because A' is a solution of the MOMD obtained from the original Steiner tree problem, all vertices from S are included in some arc from A' and hence they are part of tree T. Therefore T is a solution to the original Steiner tree problem.

In summary, the problem of finding a Steiner tree of cost at most k can be solved by converting the undirected graph G and the set S of vertices to a directed graph $\hat{G}$ and the set P of packages, finding a solution of the corresponding MOMD problem with the cost at most k, and converting the solution to the Steiner tree. $\square$

Theorem 2. *The decision variant of the multiple-origin-multiple-destination problem is an NP-complete problem.*

Proof. The problem of finding a solution of cost at most k of the MOMD problem belongs to NP. The NP-complete problem of finding a Steiner tree of cost at most k can be reduced to the problem of finding a solution of cost at most k of the MOMD problem. Hence, the decision variant of the MOMD problem is NP-complete. $\square$

SAT Models

We describe two constraint models for the MOMD problem that can be used to find a solution with cost restricted by lower and upper bounds. These models will then be used to find an optimal solution using the dichotomic branch-and-bound algorithm. We will describe the constraint model using arithmetic constraints over Boolean variables and this model is then translated to a SAT formula.

The Flow Model

The MOMD problem requires that for each package there exists a path from the package's origin to the package's destination. The *Flow model* describes a path separately for each package, accumulates all the arcs used by the packages, and restricts the total cost.

Let $G = (V, E, w)$ be a directed arc-weighted graph and P be a set of packages. For each package $p \in P$ and for each arc $a \in E$ we introduce a Boolean decision variable $\mathit{Used}[a, p]$ that indicates whether or not arc a is used to transport package p. For each package $p \in P$ and for each vertex $x \in V$ a Boolean variable $\mathit{Flow}[x, p]$ indicates whether or not the transport of package p goes through the vertex x.

Let $\mathit{InArcs}(x)$ be the set of incoming arcs to x and $\mathit{OutArcs}(x)$ be the set of outgoing arcs from x. Formally,

$$\mathit{InArcs}(x) = \{(y, x) \mid (y, x) \in E\},$$
$$\mathit{OutArcs}(x) = \{(x, y) \mid (x, y) \in E\}$$

To model a transport path for a package we specify the flow preservation constraints. These constraints describe that each package must leave its origin and must arrive at its destination, and if the package goes through some vertex then it must enter the vertex and leave it (both exactly once). In the case of origin, the package only leaves it and, similarly, in the case of destination, the package only enters it. Formally, for each package $p \in P$ we introduce the following flow preservation constraints (recall that domains of all the variables are Boolean, that is, $\{0, 1\}$):

$$\forall a \in \mathit{InArcs}(\mathit{orig}(p)) : \mathit{Used}[a, p] = 0$$
$$\forall a \in \mathit{OutArcs}(\mathit{dest}(p)) : \mathit{Used}[a, p] = 0$$
$$\mathit{Flow}[\mathit{orig}(p), p] = 1$$
$$\mathit{Flow}[\mathit{dest}(p), p] = 1$$
$$\forall x \in V \setminus \{\mathit{orig}(p)\} : \sum_{a \in \mathit{InArcs}(x)} \mathit{Used}[a, p] = \mathit{Flow}[x, p]$$
$$\forall x \in V \setminus \{\mathit{dest}(p)\} : \sum_{a \in \mathit{OutArcs}(x)} \mathit{Used}[a, p] = \mathit{Flow}[x, p]$$

We use Boolean decision variables $\mathit{Used}[a]$ to describe whether or not a given arc $a \in E$ is used by any package. This is modeled using the following constraint:

$$\max_{p \in P} \mathit{Used}[a, p] = \mathit{Used}[a]$$

The objective is then expressed using the constraint:

$$\mathit{Obj} = \sum_{a \in E} \mathit{Used}[a] \times w(a) \tag{1}$$

where Obj is a variable describing the total cost of solution.

Notice that the size of the model depends on the number of packages, vertices, and arcs so for sparse graphs the model is smaller than for dense graphs. More precisely, the number of decision variables is $nq + eq + e$, where $n = |V|, e = |E|, q = |P|$ and all the decision variables are Boolean.

The Reachability Model

The Flow model finds a path for each package explicitly. As some packages might share parts of their paths, it might be beneficial to find a path between vertices just once and then for each package ensuring that a path exists. This idea is behind our *Reachability model* that decides for any pair of vertices if a path exists between them (using the selected arcs only). We will assume the same input G and P as described above.

We propose a model where for each pair of vertices the model describes whether or not a path exists between these vertices. The model basically mimics the Floyd-Warshall algorithm [2]. Again, we use Boolean decision variables $\mathit{Used}[a]$ to indicate whether or not a given arc $a \in E$ is selected in the solution. Assume that the vertices are indexed (totally ordered) by numbers from the set $\{1, 2, \ldots, |V|\}$ (when using "the node x" we will mean its index). The Boolean decision variables $\mathit{Reach}[x, y, z]$ describe whether or not there exists a path from x to y using only vertices i such that $i \leq z$. In particular $\mathit{Reach}[x, y, 0]$ says that an arc from x to y exists and is used in the solution or $x = y$. The variables are connected using the following constraints:

$$\forall x, y \in V : \mathit{Reach}[x, y, 0] = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } (x, y) \notin E, \\ \mathit{Used}[a] & \text{if } a = (x, y) \in E. \end{cases}$$

$$\begin{aligned} &\forall x, y, z \in V : \\ &\quad \mathit{Reach}[x, y, z] = \mathit{max}(\mathit{Reach}[x, y, z-1], \\ &\qquad\qquad \mathit{Reach}[x, z, z-1] \times \mathit{Reach}[z, y, z-1]) \end{aligned}$$

Now, for each package p we require that a path from $orig(p)$ to $dest(p)$ exists:

$$\forall p \in P : Reach[orig(p), dest(p), |V|] = 1$$

The objective function is expressed using the constraint (1) as in the Flow model. Now, the number of Boolean decision variables is $(n+1)n^2 + e$. Notice that the number does not depend on the number of packages.

Optimization Procedure

The constraint models specified in the previous section describe the decision variant of the MOMD problem – the value k from the problem specification can be used as the upper bound for the objective variable Obj using the constraint $Obj \leq k$. To solve the minimization problem, we use the dichotomic version of the branch-and-bound algorithm [5], where the lower and upper bounds are moving closer to each other by splitting the interval between them into halves until the bounds become equal. Let $Bound^-$ be the known lower bound of the objective function and $Bound^+$ be the known upper bound of the objective function. Then the dichotomic branch-and-bound algorithm works as follows. It finds a middle value $Bound$ between $Bound^-$ and $Bound^+$ and tries to find a solution better or equal to $Bound$. If no solution exists then the lower bound is increased to $Bound + 1$. If a solution is found then the upper bound is decreased to $Bound$. This process is repeated until $Bound^- = Bound^+$. The pseudocode is shown in Algorithm 1.

```
repeat
    $Bound \leftarrow round((Bound^+ + Bound^-)/2)$
    $Sol \leftarrow Solve(Cons \cup \{Obj \leq Bound\}$
    if $Sol$=$fail$ then
        $Bound^- \leftarrow Bound + 1$
    else
        $Bound^+ \leftarrow Bound$
    end
until $Bound^- = Bound^+$;
```

Algorithm 1: A dichotomic version of branch-and-bound.

We calculated the initial bounds as follows. The lower bound is the maximum from the costs of shortest paths for all the packages. Obviously, no better solution exists as for each package we need to go from its origin to its destination. The upper bound is the cost of the straightforward solution – the shortest (cost optimal) path is found for each package and the solution is defined as all the arcs in the union of these shortest paths.

Translation to SAT

The proposed models are translated to SAT formulas. We use the Picat SAT compiler, which employs hybrid encodings for constraints [11]. The following summarizes how the Boolean constraints used in the models are translated to SAT:

$$\begin{aligned}
&max(\{X_1, X_2, \ldots, X_n\} = Y :\\
&\quad Y = 1 \Rightarrow X_1 \vee X_2 \vee \cdots \vee X_n\\
&\quad Y = 0 \Rightarrow \neg X_1 \wedge \neg X_2 \wedge \cdots \wedge \neg X_n\\
&sum(\{X_1, X_2, \ldots, X_n\} = Y :\\
&\quad Y = 1 \Rightarrow exactly\_one(\{X_1, X_2, \ldots, X_n\})\\
&\quad Y = 0 \Rightarrow \neg X_1 \wedge \neg X_2 \wedge \cdots \wedge \neg X_n\\
&exactly\_one(\{X_1, X_2, \ldots, X_n\}) \Leftrightarrow\\
&\quad at\_most\_one(\{X_1, X_2, \ldots, X_n\}) \wedge\\
&\quad at\_least\_one(\{X_1, X_2, \ldots, X_n\})
\end{aligned}$$

The at_most_one constraint $\Sigma_i X_i \leq 1$ is encoded into CNF by using Chen's algorithm [1], which splits the sequence of Boolean variables into two subsequences, and encodes the sum $\Sigma_i^n X_i$ as the Cartesian product of the two subsequences. The objective constraint is broken down to primitive constraints, which are encoded as adders and multipliers.

ASP Model

We have encoded the MOMD problem in Answer Set Programming (ASP). As usual in ASP the coding is very concise and, basically, based on a generate & test programming scheme. Assume the input graph is encoded by a predicate road/3 where road(a, b, w) denotes that the arc (a, b) has cost w. Each origin/destination (o/d) pair is imposed by a binary predicate trip. We can define the predicate node by projection. Then, using a choice rule we can either select an edge or not. Reachability is computed on the subgraph of the selected nodes, and a constraint is added to force the connectivity between all o/d pairs. Finally, the cost optimization is imposed.

```
node(A) :- road(A,_,_).
node(B) :- road(_,B,_).

{selected(A,B,C)} :- road(A,B,C).

reach(A,A) :-  node(A).
reach(A,B) :-  selected(A,C,_),  reach(C,B).

:- trip(Origin,Destination),
    not reach(Origin,Destination).
cost(Cost) :-  Cost = #sum{C,A,B : selected(A,B,C)}.
#minimize {C:cost(C)}.
```

Minimum path for each o/d pair in the complete graph can be modeled with few lines of code in order to compute bounds that can be used in the successive search. The obtained code however suffers from huge grounding and the running time is (slightly) better than the one of the above code only in few instances.

MiniZinc Model

We have also tested a similar, highly non-deterministic, encoding in MiniZinc [6]. Basically, for each o/d pair (assume there are p of them) we introduce an array path$[i]$ where $i = 1, \ldots, p$ aimed at storing the path using selected edges. These arrays have length n (number of nodes). path$[i, j]$ is the j-th node found in the path that leads the i-th origin to the i-th destination. When the destination d is reached, all successive values of the vector are d. No other "loops" are possible.

We found also convenient to use an auxiliary `successor` predicate that allows to restrict the domain of the next element of the path.

```
% domain for the successor (for the path p)
constraint
  forall (i in 1..nodes, p in 1..pairs,
          j in 1..nodes where i != trip[p,2])
       (graph[i,j]=0 -> successor[p,i] != j);

% the target has no successor (self-loop)
constraint
  forall (p in 1..pairs)
     (successor[p,trip[p,2]]=trip[p,2]);

% Every path starts from the source
constraint
  forall (p in 1..pairs)
     (path[p, 1] = trip[p, 1]);

% the target is eventually reached
constraint
  forall (p in 1..pairs)
     (path[p,nodes]=trip[p,2]);

% Check/Force that the path is feasible
%   (use successor)
constraint
  forall (p in 1..pairs, i in 2..nodes)
    (path[p, i] = successor[p, path[p,i-1]]);

% No self-loops during search
constraint
  forall (p in 1..pairs, i in 1..nodes - 1)
    (path[p,i+1] = path[p,i] ->
        path[p,i] = trip[p,2]);

% Connection between selected and path
constraint
  forall (p in 1..pairs,  i in 2..nodes)
    (path[p, i] > 0 ->
        selected[path[p, i - 1], path[p, i]]);

% Cost
constraint
  cost = sum (i in 1..nodes, j in 1..nodes)(
    selected[i, j] * graph[i, j]
  );
```

The running times are very unsatisfactory. We experimentally found the best results with this program designed search heuristics

```
solve::int_search([path[p, i]| p in 1..pairs,
                               i in 1..nodes],
           first_fail, indomain_min, complete)
           minimize cost;
```

Experimental Evaluation

We have implemented the SAT models in Picat [8], which employs Lingeling as the SAT solver. In Picat, it is also possible to use CP and MIP solvers for the same models, but the SAT solver overwhelmingly outperforms the CP and MIP solvers for these models. We used Picat 1.9b1 running with MacOS X 10.11.4 on 1.7GHz Intel Core I7 with 8GB 1600 MHz DDR3 RAM. For this comparison we used the instances of the Transport domain from the optimal track of the International Planning Competition 2014 [4]. Table 1 gives characteristics, including the lower bound, the upper bound, and the optimal solution, of each of the instances.

Table 1 also shows the runtimes (in seconds) for both SAT models to find and proof optimal solutions. The Flow model is clearly faster on all the benchmark instances. This is probably due to the fact that the Flow model uses fewer decision variables than the Reachability model for the instances.

We also developed models in ASP and MiniZinc for the problem, both of which use reachability constraints. The clingo solver [7] with the default setting found optimal solutions for 19 of the 20 instances, 12 of which were found within 1 minute each, but took considerably more time than our models on the solved instances, and failed to solve instance p17 within 24 hours. Although both ASP model and our SAT models use SAT, our models are significantly more efficient than ASP because of the compact and efficient encodings used for the constraints.

We also compared our models with a model implemented in MiniZinc [6], which uses constraints to prevent cycles. This model, when run by Gecode, solved 10 of the 20 instances, 7 of which were solved within 1 minute each, but failed to solve 10 of the instances under the time limit of 24 hours per instance. This comparison once again demonstrates the effectiveness of the SAT models for the problem.

Conclusions

The paper proposes a path finding problem called multiple-origin-multiple-destination (MOMD) problem, where the objective is minimizing the total cost of used arcs. This problem is motivated by transportation and network problems where the variation between paths should be minimized. We showed that this problem is NP-complete, and proposed two SAT-based models to solve the problem optimally. The model based on flow-preserving constraints seems computationally more efficient than the model based on reachability constraints. The initial experiments also showed that the SAT model is more efficient than the CP and ASP models.

There are several open problems to be resolved. First, it would be interesting to compare both SAT models using problems with different numbers of packages, in particular because the number of decision variables in the Flow model depends on the number of packages while the number of variables in the Reachability model is independent of the number of packages. Second, it would be interesting to study better lower bounds for the objective function, in particular in relation to the original motivation of computing better heuristic estimates for the Transport problem. Third, it would be interesting to compare efficiency of the SAT solvers with state-of-the-art MIP solvers, in particular for the Flow model that uses linear constraints.

Acknowledgments.

Roman Barták is supported by the Czech Science Foundation under the project P103-15-19877S.

Table 1: Problem instances and solution times (proof of optimality).

	Problem characteristics						Runtime in seconds	
instance	#vertices	#arcs	#packs	lb	ub	opt	Flow model	Reach model
p01	5	12	3	58	122	122	**0.096**	0.183
p02	10	34	4	90	189	162	**0.395**	0.669
p03	12	40	5	122	235	234	**0.568**	1.402
p04	15	44	5	86	197	197	**0.599**	1.913
p05	18	70	6	130	408	284	**6.868**	6.980
p06	20	62	6	134	408	355	**1.330**	5.013
p07	10	24	3	307	367	352	**0.285**	0.554
p08	20	62	4	339	391	362	**0.697**	2.271
p09	24	84	5	316	440	384	**1.366**	5.913
p10	30	94	5	334	510	507	**2.093**	7.101
p11	36	144	6	294	539	529	**21.283**	30.676
p12	40	138	6	376	603	558	**5.634**	18.663
p13	15	40	4	263	499	491	**0.673**	2.642
p14	30	98	4	288	376	376	**3.796**	7.252
p15	36	124	5	278	693	646	**4.159**	17.145
p16	45	148	5	310	841	688	**4.159**	18.554
p17	54	204	6	318	1153	832	**31.884**	47.948
p18	60	222	6	356	902	890	**15.922**	46.539
p19	60	222	7	373	966	911	**22.287**	56.813
p20	60	222	7	373	966	911	**22.212**	46.377

References

[1] Chen J.: A new SAT encoding of the at-most-one constraint. *Proc. of the 9th Int. Workshop of Constraint Modeling and Reformulation*, 2010.

[2] Floyd R. W.: Algorithm 97: Shortest Path. *Communications of the ACM* 5 (6): 345, 1962.

[3] Garey M. R.; Johnson D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[4] International Planing Competions web site, *http://ipc.icaps-conference.org/*, Accessed March 24, 2016.

[5] Land A. H. and Doig A. G.: An automatic method of solving discrete programming problems. *Econometrica* 28(3): 497–520, 1960.

[6] MiniZinc web site, *http://www.minizinc.org*, Accessed April 22, 2016.

[7] Potassco, the Potsdam Answer Set Solving Collection web site, *http://potassco.sourceforge.net*, Accessed April 22, 2016.

[8] Picat web site, *http://picat-lang.org/*, Accessed March 24, 2016.

[9] Surynek P.: Compact Representations of Cooperative Path-Finding as SAT Based on Matchings in Bipartite Graphs, *Proceedings of the 26th International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, IEEE, pp. 875–882, 2014.

[10] Thomas R. S. D. and Wells J. M.: Multiple-Origin Single-Destination Transit Routing. *Interfaces* 10(2):41–43, 1980.

[11] Zhou N.-F., Kjellerstrand H.: The Picat-SAT Compiler. *Practical Aspects of Declarative Languages*, LNCS 9585, pp. 48–62 , 2016.