CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Branch-and-Cut Algorithm for the Non-Preemptive Capacitated Swapping Problem

Güneş Erdoğan
Jean-François Cordeau
Gilbert Laporte

September 2008

CIRRELT-2008-44

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · UQÀM · HEC Montréal · École Polytechnique Montréal · Université de Montréal

A Branch-and-Cut Algorithm for the Non-Preemptive Capacitated Swapping Problem

Güneş Erdoğan[1,2,3], Jean-François Cordeau[1,2], Gilbert Laporte[1,3,*]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

[3] Canada Research Chair in Distribution Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. This paper models and solves a capacitated version of the Non-Preemptive Swapping Problem. This problem is defined on a complete digraph G = (V, A), at every vertex of which there may be one unit of supply of an item, one unit of demand, or both. The objective is to determine a minimum cost capacitated vehicle route to transport the items in such a way that all demands are satisfied. The vehicle can carry more than one item at a time. Three integer programming formulations of the problem are provided. Several classes of valid inequalities are derived and incorporated within a branch-and-cut algorithm, and extensive computational experiments are performed on instances adapted from TSPLIB.

Keywords. Swapping problem, robot arm travel, non-preemptive, capacitated, integer programming, branch-and-cut.

Acknowledgements. This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 227837-04 and 39682-05. This support is gratefully acknowledged.

---

* Corresponding author: Gilbert.Laporte@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2008

1 Introduction

This paper proposes a branch-and-cut algorithm for the *Non-Preemptive Capacitated Swapping Problem* (NCSP). In this problem, we are given a complete directed graph $G = (V, A)$, where V is the set of vertices and A is the set of arcs. Vertex 0 is called the *depot*. Every vertex $i \in V$ may supply one item of type a_i, and demand one item of type b_i. We assume that except for the depot every vertex supplies or demands an item. Let $T \subset V$ be the set of *transshipment vertices*, i.e., vertices i for which $a_i = b_i$. The items are unsplittable and undroppable at intermediate vertices. The item types constitute the set K, where $|K| \leq |V|$. Every item of type k has a weight q_k. The total supply and the total demand for each item type are equal. The cost of traveling from vertex i to vertex j is denoted by c_{ij} and satisfies the triangle inequality.

In the NCSP a vehicle of capacity Q must satisfy all demand by transporting the items, using a minimum cost route starting and ending at the depot. Traversing an arc without transporting an item is called *deadheading*. As in Anily and Hassin (1992), an *optimal* solution is one that uses the least number of arcs among the solutions with lowest cost. Figure 1 depicts an instance of the NCSP. The items supplied and demanded by each vertex $i \in V$ are denoted by the pair (a_i, b_i). The optimal solution consists of traversing the left and right triangles, in the order $(0, 1, 2, 0, 3, 4, 0)$, starting with an empty vehicle.

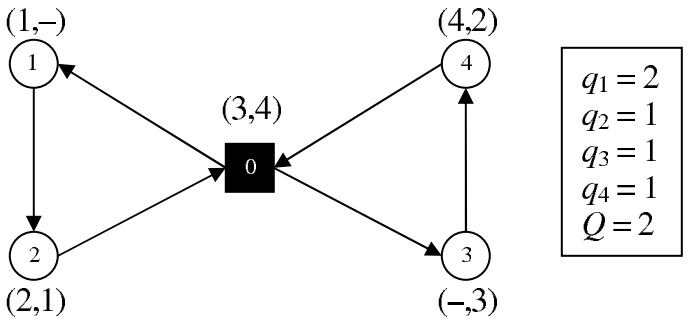

Figure 1: A sample instance of the NCSP

The NCSP is a generalization of the *Swapping Problem* (SP), introduced by Anily and Hassin (1992). In the SP, $q_k = Q$ for all $k \in K$. The well-known *Stacker Crane Problem* (Frederickson et al. 1978) is a special case of the SP where each

item type is supplied and demanded by exactly one vertex. An instance of the SP containing both *droppable* and *undroppable* item types is called *mixed*. The variant of the SP in which all the item types are droppable is called the *Preemptive Swapping Problem* (PSP), and the variant in which all the item types are undroppable is called the *Non-Preemptive Swapping Problem* (NSP). Anily and Hassin (1992) proved that the SP, PSP, and NSP are NP-hard and presented two polynomial time approximation algorithms with a worst-case performance ratio of 2.5. Anily et al. (1999) have studied the special case of the SP on a line, provided structural properties of optimal solutions, and presented an optimal $O(n^2)$ algorithm. In subsequent work Anily et al. (2008) have studied the PSP on a tree. This variant was shown to be NP-hard in general, and polynomially solvable for a single item type. The authors have provided a heuristic with a performance guarantee of 1.5. They also constructed a table of the previously studied variants of the SP, together with their corresponding complexity status.

The studies mentioned up to this point have been purely theoretical. In a recent paper by Bordenave et al. (2008b), a computational study for the NSP was conducted for the first time. The authors first presented an integer programming formulation that exploits the structure of the optimal solutions. Using a branch-and-cut algorithm, they were able to solve instances containing up to 200 vertices and eight item types. The same authors (Bordenave et al. 2008c) also developed a branch-and-cut algorithm for the PSP and were able to solve instances containing up to 120 vertices and eight item types. In a third study (Bordenave et al. 2008a), constructive and improvement heuristics were developed for the mixed SP. Instances containing up to 10,000 vertices and eight item types were successfully solved within approximately one percent of optimality.

A problem closely related to the SP is the *One-Commodity Pickup-and-Delivery Traveling Salesman Problem* (1-PDTSP). The 1-PDTSP consists of determining a Hamiltonian cycle for a capacitated vehicle making pickups and deliveries at the customer vertices. Three differences from the NCSP are 1) in the 1-PDTSP there is only one item type; 2) the demand and supply quantites at a vertex may be more than one; 3) there are no transshipment vertices. Hernández-Pérez and Salazar-González (2004) have studied the 1-PDTSP and proposed an integer programming formulation. They have also devised two sets of valid inequalities based on Benders decomposition and the set-packing structure embedded within the problem. Using a branch-and-cut algorithm, they were able to solve instances with up to 50 customers. In subsequent work (Hernández-Pérez and Salazar-González 2007), the authors have adapted inequalities from the *Stable Set Problem* and the *Capacitated Vehicle Rout-*

ing Problem and refined their algorithm. They were thus able to solve instances with up to 100 customers. Because the structure of the NCSP resembles that of $|K|$ 1-PDTSPs superimposed upon each other, valid inequalities for the 1-PDTSP can be used for every item type, and for the union of all item types into a single one. The authors have recently studied the *Multi-commodity one-to-one Pickup-and-Delivery Traveling Salesman Problem* (m-PDTSP) in which every item type has a unique supply and demand point (Hernández-Pérez and Salazar-González 2008). Three algorithms were devised based on different formulations and decomposition techniques. The largest instances they were able to solve contains 15 commodities. This problem is similar to the NCSP with three differences: 1) in the m-PDTSP, every item type has a unique supply and demand point; 2) a vertex can be the supply or demand point for more than one item type; and 3) there are no transshipment vertices.

The applications for the NCSP are similar to those of the SP. Examples for the SP arise in the area of robotics, like the problem of relocating m objects among n stations (Atallah and Kosaraju 1988, Frederickson and Guan 1992, 1993, Anily et al. 1999, Chalasani and Motwani 1993). In these references, a robotic arm can carry one item at a time. The NCSP relaxes the unit capacity constraint, consequently increasing the flexibility. Another application exists in the manufacturing context, for the retooling of an assembly line to manufacture a different product (Escudero 1988). A vehicle that can carry more than one workstation at a time must reorder the workstations in order to minimize the setup time. A third application is the optimal sequencing of pick-and-place operations in the manufacturing operations of printed circuit boards (Ball and Magazine 1988).

Our aim is to formulate the NCSP and solve it to optimality by branch-and-cut. The remainder of the paper is organized as follows. In Section 2, we provide further definitions and results for the NCSP. In Section 3, we propose a graph transformation and preprocessing rules to simplify the instance at hand. Three integer linear programming formulations are developed in Section 4. In Section 5, we present valid inequalities for these formulations. A heuristic and a branch-and-cut algorithm are developed in Section 6. Section 7 contains the results of our computational experiments. Conclusions follow in Section 8.

2 Solution Properties

In this section we prove three properties of the NCSP.

Proposition 1: The NCSP is NP-hard.

Proof: The result follows from the fact that the NCSP includes the NSP as a special case with $q_k = Q = 1, \forall k \in K$. □

Proposition 2: There exist NCSP instances in which transshipment vertices are visited in an optimal solution.

Proof: The example provided by Bordenave et al. (2008b) demonstrates that transshipment vertices may actually be visited in an optimal solution of an NSP instance, which extends to the NCSP. □

Proposition 3: All transshipment vertices except the depot either have a degree of zero or four in an optimal solution.

Proof: This statement has been proven for the NSP in Lemma 5 of Bordenave et al. (2008b). The result relies on the fact that a single visit to a transshipment vertex does not make any change in the composition of the items being carried by the vehicle. Furthermore, such a visit may increase the length of the route because of the triangle inequality, as well as the number of arcs traversed. So if a transshipment vertex is visited in an optimal solution, it must be visited twice. Since the arguments do not use any information regarding vehicle capacity, the result is also valid for NCSP. □

Proposition 4: All vertices in V except the depot have a degree of zero, two, or four in an optimal solution of an instance of the NCSP.

Proof: Since the items are not allowed to be dropped in intermediate vertices, the vehicle can either visit a vertex to pickup the supply, or to deliver the demand, or execute both operations simultaneously. If the pickup and delivery operations are performed on separate visits the vertex will have a degree of four. If they are performed simultaneously in a single visit, the vertex will have a degree of two. Including the transshipment vertices, which may have a degree of zero or four yields the result. □

Proposition 5: The depot may have a degree of two, four, or six in an optimal solution of an instance of the NCSP.

Proof: In addition to possible pickup and delivery operations, the depot is also the designated location for the start and the end of the route. If the route starts without a pickup operation, ends without a delivery operation, and the pickup and delivery operations are executed on separate visits, the depot may have a degree of six. If at least one of the pickup and delivery operations coincides with the start or the end of the route, the depot will have a degree of four. If both operations coincide with the start and the end of the route, or the depot does not involve any supply or demand, then the degree of the depot will be two. □

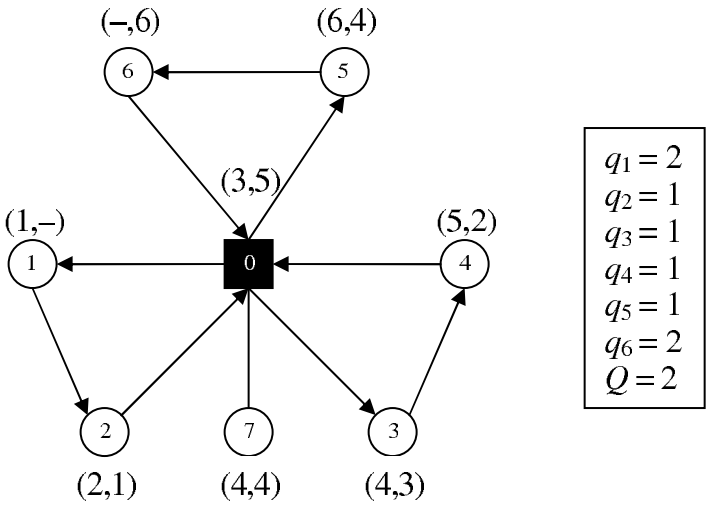

Figure 2: Example of an optimal solution in which the degree of the depot is six

For an example of an instance for which the depot has a degree of six, consider the instance of NCSP depicted in Figure 2. The cost of every arc is equal to one. Note that the cost of traversing a triangle is three, whereas visiting the vertices in the triangle separately generates a cost of four. All vertices on the triangles must be visited since they are not elements of T. Hence, a valid lower bound on the optimal solution value, computed by traversing three triangles is 9. Let p_k and d_k denote the operations of picking up item k and delivering item k, respectively. The following feasible solution has a cost of 9: $(p_1, d_1, p_2, p_3, d_3, p_4, d_2, p_5, d_5, d_4, p_6, d_6)$. The actual route is given by: $(0, 1, 2, 0, 3, 4, 0, 5, 6, 0)$. This solution consists of traversing the bottom left, bottom right, and top triangles, in that order. Any solution that does not traverse the triangles is bound to have a higher cost. The reader can check that traversing the triangles in any other order requires extra visits to certain vertices. Hence, the solution given above is the unique optimum. In this solution, both the first and the last move are deadheadings from and to the depot, respectively. Combined with the pickup and delivery operations executed on different visits, the depot has a degree of six. Note that for the optimal solution of the instance depicted in Figure 1, the depot has a degree of four, which can be verified in a similar way.

3 Operation Digraph

In this section we construct an auxiliary graph which we subsequently use to develop integer programming formulations.

3.1 Construction

Integer programming formulations for classical routing problems seek to construct Hamiltonian tours. They typically contain degree constraints, subtour elimination constraints and integrality constraints. However, swapping problems sometimes require a vertex to be visited more than once, which implies that the solution may contain subtours. In order to exploit the structure of classical formulations for routing problems, we will first construct a directed *operation digraph* $\bar{G}$ on which we can emulate the multiple visits using simple paths. On this graph, we will construct a circuit that describes the route as well as the pickup and delivery operations of the vehicle, which we call the *main path*. Note that an optimal solution may contain certain arcs that are not connected to the main path. These arcs correspond to unvisited transshipment vertices.

For a given instance of the NCSP defined on a complete directed graph $G = (V, A)$, we construct the operation digraph $\bar{G} = (\bar{V}, \bar{A})$. The vertex set $\bar{V}$ is made up of four disjoint vertex sets: O consists of a copy of the depot denoting the start of the route, P is the set of vertices supplying an item, D is the set of vertices demanding an item, and finally R consists of another copy of the depot denoting the end of the route. Thus $\bar{V} = O \cup P \cup D \cup R$. Vertex sets O and R are included in order to handle possible deadheadings from and to the depot, respectively. If a vertex in V supplies an item, we insert a copy of this vertex in P. Likewise, if a vertex in V demands an item, we insert a copy of this vertex in D. Note that if a vertex other than the depot both supplies and demands items, there will be two copies of this vertex in $\bar{V}$, and if the depot involves a supply and a demand, there will be four copies of the depot in $\bar{V}$. We call the vertices of V the *parent vertices*, and the vertices of $\bar{V}$ the *child vertices*. If a child vertex is in P we call it a *pickup child vertex*, and if it is in D, we call it a *delivery child vertex*. We denote $\pi(i)$ the parent of a vertex $i \in \bar{V}$. Visiting a pickup (delivery) child vertex corresponds to performing a pickup (delivery) operation at the parent vertex.

The arc set $\bar{A}$ is constructed as follows. We first introduce arcs from the copy of the depot in O to every element of P, and from every element of D to the copy of the depot in R. These arcs describe the start and the end of the tour. We then add

arcs (i, j) from every vertex $i, j \in P \cup D$ such that $\pi(i) \neq \pi(j)$. These arcs describe the movements of the vehicle from $\pi(i)$ to $\pi(j)$ in G. For vertices $i \in D$ and $j \in P$ with $\pi(i) = \pi(j)$ and $\pi(i) \notin T$ we create an arc from i to j. Using such an arc on the main path means that simultaneous delivery and pickup operations are taking place in a single visit at $\pi(i)$ in G. The reverse of these arcs are not defined in order to avoid symmetries in the case of simultaneous pickup and delivery operations at $\pi(i)$. For vertices $i \in P$ and $j \in D$ with $\pi(i) = \pi(j)$ and $\pi(i) \in T$ we create an arc from i to j. Using such an arc means the parent node is not being visited by the vehicle. Again, the reverse of this arc is intentionally avoided to break the symmetry. Let $\bar{A}_T$ denote the arcs connecting the sibling vertices in $P \cup D$, the parents of which are transshipment vertices. With this construction a circuit on $\bar{G}$ describes a route on G. The operation digraph of the optimal solution of the instance shown in Figure 2 is given in Figure 3. The parent vertex indices are indicated on the child vertices, and the item type supplied or demanded by the parent vertex is denoted in brackets.

To avoid confusion with the costs associated with the original graph G, we define the cost of traversing arc (i, j) in $\bar{A}$ as $\bar{c}_{ij}$. We set $\bar{c}_{ij}$ in a way to reflect the definition of an optimal solution, which effectively defines a lexicographic relationship between two objectives. The first objective is to minimize the solution cost, whereas the second objective is to minimize the number of arcs in the solution. To this end, we use the scaling approach of Sherali (1982) for lexicographic multiobjective optimization. To achieve the right ordering of objectives, we multiply the first objective by a strict upper bound on the second one, and add it to the second one. By Propositions 4 and 5, we know that in an optimal solution the total degree of vertices in G is bounded above by $4|V| + 2$, and we therefore set an upper bound of $2|V| + 1$ on the number of arcs in the solution. We increment this value by one to have a strict upper bound and we set the arc lenghts in the operation digraph as $\bar{c}_{ij} = (2|V| + 2)c_{\pi(i)\pi(j)} + 1$ for $(i, j) \in \bar{A}_1$, and $\bar{c}_{ij} = 0$ for $(i, j) \in \bar{A} \setminus \bar{A}_1$.

3.2 Simplifications

Although the graph transformation presented above makes modeling easier, it also increases the number of vertices by a factor of 2, consequently increasing the complexity. Now we turn our attention to simplifying the problem instance at hand. Our first step is to reset the vehicle capacity to its maximum usable capacity. To determine the maximum usable capacity of a vehicle, we solve a knapsack problem with all the items and their profits being equal to their weights. If the optimal solution value of this problem is less than the vehicle capacity, we set the value of the

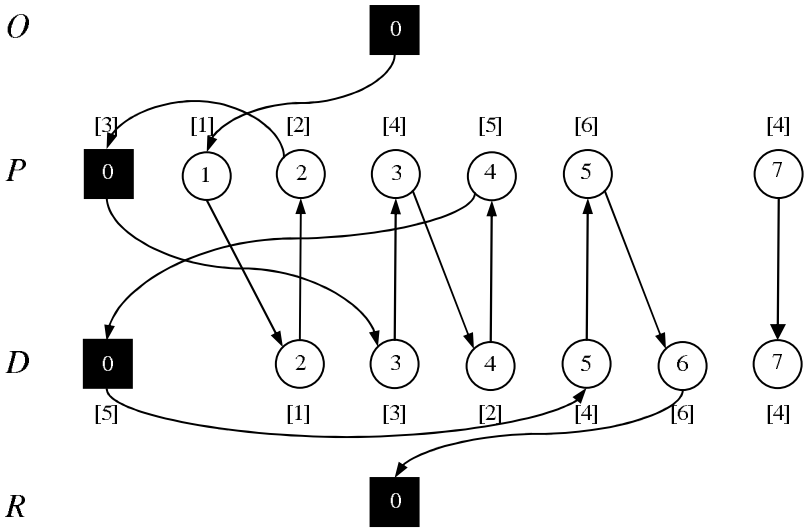

Figure 3: Optimal solution of the instance of Figure 2 depicted on the corresponding operation digraph

vehicle capacity to be equal to the optimal objective value. As an example, consider the case where there are three item types, with $q_1 = 3$, $q_2 = 4$, and $q_3 = 5$. Also assume that there are one item of type 1, two items of type 2, and two items of type 3. Note that a vehicle with a capacity of 6 will never be used completely, hence its capacity can be reset to 5. Likewise, a vehicle capacity 15 is equivalent to a vehicle of capacity 14.

The operation digraph described above can be simplified by eliminating arcs (i, j) whose use would make the weight of items being carried on-board exceed the vehicle capacity. There are three such types of arcs:

1) $i \in P, j \in P, q_{a_{\pi(i)}} + q_{a_{\pi(j)}} > Q$

2) $i \in D, j \in D, q_{b_{\pi(i)}} + q_{b_{\pi(j)}} > Q$

3) $i \in P, j \in D, a_{\pi(i)} \neq b_{\pi(j)}, q_{a_{\pi(i)}} + q_{b_{\pi(j)}} > Q$.

Using an arc of the first type implies that the vehicle picks up an item of type $a_{\pi(j)}$ while transporting an item of type $a_{\pi(i)}$, and the sum of the item weights exceeds the vehicle capacity. Using an arc of the second type implies that the vehicle delivers two items consecutively, and the sum of the item weights exceed the vehicle capacity. Finally, using an arc of the third type implies that the vehicle is

delivering up an item of type $b_{\pi(j)}$ while transporting an item of type $a_{\pi(i)}$, where the items are different and the sum of their weights exceeds the vehicle capacity.

These three preprocessing rules have a direct consequence. Given an instance of the NCSP with the weight of every item type greater than half the vehicle capacity, i.e. $q_k > Q/2$, $\forall k \in K$, no two items will fit together in the vehicle. Consequently, all arcs between pickup child vertices and all arcs between delivery child vertices will be eliminated. Hence this instance can be solved as an instance of the NSP, which has a much better computational performance. The previous example with a vehicle of capacity 6, when the two preprocessing rules are applied sequentially, becomes an instance of the NSP.

Searching for arcs that exceed the vehicle capacity raises the question of the extent to which the vehicle capacity can be used for the rest of the arcs. Note that if the vehicle enters a pickup child vertex, there must be enough capacity to load the associated item, so the inflow capacity on the associated arc should be limited accordingly. Likewise, if the vehicle leaves a delivery child vertex, on the corresponding arc the vehicle can transport at most its total capacity minus the weight of the item just delivered. Using these two facts, we define the capacity of each arc as:

$$Q_{ij} = \begin{cases} Q - q_{a_{\pi(j)}} & i \in P, j \in P \\ Q & i \in P, j \in D \\ \min\left\{Q - q_{a_{\pi(j)}}, Q - q_{b_{\pi(i)}}\right\} & i \in D, j \in P \\ Q - q_{b_{\pi(i)}} & i \in D, j \in D. \end{cases} \tag{1}$$

Although this modification has no effect on the operation digraph, a tighter value for vehicle capacity is observed to be helpful in the integer programming models presented in the next section.

4 Integer Programming Formulations

Using the operation digraph, we provide three integer programming models for the NCSP.

4.1 Directed multicommodity flow formulation

Our first formulation uses flow variables for each item type. We first define the parameter

$$\alpha_{ik} = \begin{cases} 1 & i \in P \text{ and } a_{\pi(i)} = k \\ -1 & i \in D \text{ and } b_{\pi(i)} = k \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

We denote by P_T the subset of vertices of P whose parents are elements of T, and by D_T the subset of vertices of D whose parents are elements of T. For each subset $S \subset \bar{V}$, let $\delta^+(S) = \{(i,j) \in \bar{A} : i \in S, j \notin S\}$, and $\delta^-(S) = \{(i,j) \in \bar{A} : i \notin S, j \in S\}$. For the sake of simplicity we write $\delta^+(i)$ and $\delta^-(i)$ instead of $\delta^+(\{i\})$ and $\delta^-(\{i\})$. We also define $x(A') := \sum_{(i,j)\in A'} x_{ij}$. Let x_{ij} be equal to 1 if the vehicle traverses arc $(i,j) \in \bar{A}$ and 0 otherwise, and let y_{ijk} be the number of units of item type k carried on the arc $(i,j) \in \bar{A}$. The formulation is as follows:

(NCSP1)

$$\text{minimize} \sum_{(i,j)\in\bar{A}} \bar{c}_{ij} x_{ij} \tag{3}$$

subject to

$$x(\delta^+(i)) + x(\delta^-(i) \cap \bar{A}_T) = 1 \quad (i \in \bar{V} \setminus R) \tag{4}$$
$$x(\delta^-(i)) + x(\delta^+(i) \cap \bar{A}_T) = 1 \quad (i \in \bar{V} \setminus O) \tag{5}$$
$$\sum x(\delta^+(S)) \geq 1 \quad (S \subset \bar{V} \setminus R : S \setminus (P_T \cup D_T) \neq \emptyset) \tag{6}$$
$$\sum_{j\in\delta^+(i)} y_{ijk} - \sum_{j\in\delta^-(i)} y_{jik} = \alpha_{ik} \quad (i \in \bar{V}, k \in K) \tag{7}$$
$$\sum_{k\in K} q_k\, y_{ijk} \leq Q_{ij}\, x_{ij} \quad ((i,j) \in \bar{A} : j \notin P) \tag{8}$$
$$y_{ijk} \geq 0 \quad ((i,j) \in \bar{A}) \tag{9}$$
$$x_{ij} = 0 \text{ or } 1 \quad ((i,j) \in \bar{A}). \tag{10}$$

Equalities (4) force an arc out of every vertex, except the delivery child vertices of transshipment parent vertices. Equalities (5) force an arc into every vertex, except the pickup child vertices of transshipment parent vertices. Inequalities (6), called the *connectivity constraints* require every vertex other than the child vertices of transshipment vertices to be reachable from the copy of the depot in O. Equalities (7) conserve the flow of items. Inequalities (8) prohibit the vehicle from exceeding its specific capacity.

4.2 Directed vehicle flow formulation

In our second formulation, we eliminate the commodity flow variables. To this end, we define the total supply of an item type k in a subset of vertices S of $\bar{V}$ as $d_k(S) = \sum_{i \in S} \alpha_{ik} q_k$, and the total net supply of items as $d(S) = \sum_{k \in K} d_k(S)$. For simplicity of notation, we write $d(i)$ instead of $d(\{i\})$. Furthermore, we define

$$r^+(S) = \begin{cases} \max\left\{0, \left\lceil \frac{d(S)}{Q} \right\rceil, \max\limits_{k \in K} \left\{ \left\lceil \frac{d_k(S)}{Q} \right\rceil \right\}\right\}, & S \subseteq (P_T \cup D_T) \text{ or } R \subseteq S \\ \max\left\{1, \left\lceil \frac{d(S)}{Q} \right\rceil, \max\limits_{k \in K} \left\{ \left\lceil \frac{d_k(S)}{Q} \right\rceil \right\}\right\}, & \text{otherwise,} \end{cases} \tag{11}$$

which denotes the minimum number of times a vehicle must leave vertex set S, and

$$r^-(S) = \begin{cases} \max\left\{0, \left\lceil \frac{-d(S)}{Q} \right\rceil, \max\limits_{k \in K} \left\{ \left\lceil \frac{-d_k(S)}{Q} \right\rceil \right\}\right\}, & S \subseteq (P_T \cup D_T) \text{ or } O \subseteq S \\ \max\left\{1, \left\lceil \frac{-d(S)}{Q} \right\rceil, \max\limits_{k \in K} \left\{ \left\lceil \frac{-d_k(S)}{Q} \right\rceil \right\}\right\}, & \text{otherwise,} \end{cases} \tag{12}$$

which denotes the minimum number of times a vehicle must enter vertex set S. The second formulation is as follows:

(NCSP2)

$$\text{minimize} \sum_{(i,j) \in \bar{A}} \bar{c}_{ij} x_{ij} \tag{13}$$

subject to

$$x(\delta^+(S)) \geq r^+(S) \quad (S \subset \bar{V}) \tag{14}$$
$$x(\delta^-(S)) \geq r^-(S) \quad (S \subset \bar{V}) \tag{15}$$

and (4), (5), (10).

Inequalities (14) and (15) are called the *rounded capacity constraints*.

4.3 Undirected vehicle flow formulation

In our third formulation, we consider the case in which the arc costs are symmetric. For this case, we construct an undirected *operation graph* by creating an edge between every pair of vertices between which an arc exists in the digraph. Let $\bar{E}$ denote the edge set of the undirected operation graph, and $\bar{E}_T$ denote the edges connecting the sibling vertices in $P \cup D$, the parents of which are transshipment vertices. Let $\delta(S) = \{(i,j) \in \bar{E} : i \in S, j \notin S\}$, with $\delta(i) \equiv \delta(\{i\})$. Let z_{ij} be a variable equal to 1 if the vehicle traverses edge $(i,j) \in \bar{E}$ and 0 otherwise. We denote the sum of the z variables within an edge set E' as $z(E') = \sum_{(i,j)\in E'} z_{ij}$. The third formulation is then

(NCSP3)

$$\text{minimize} \sum_{(i,j)\in\bar{E}} \bar{c}_{ij} z_{ij} \tag{16}$$

subject to

$$z(\delta(O)) = 1 \tag{17}$$
$$z(\delta(R)) = 1 \tag{18}$$
$$z(\delta(i) \setminus \bar{E}_T) + 2\, z(\delta(i) \cap \bar{E}_T) = 2 \quad (i \in P \cup D) \tag{19}$$
$$z(\delta(S)) \geq r^+(S) + r^-(S) \quad (S \subset \bar{V} : |S| \geq 2) \tag{20}$$
$$z_{ij} = 0 \text{ or } 1 \quad ((i,j) \in \bar{E}). \tag{21}$$

Equalities (17) and (18) set the degree of the two copies of the depot equal to 1. Equalities (19) state that the degree of a vertex in $P \cup D$ is two, except for the children of transshipment vertices which can be connected to each other and have a degree of one. Inequalities (20) eliminate subtours, as well as tours that violate the vehicle capacity.

5 Valid Inequalities

We now derive valid inequalities for our integer programming models.

5.1 Implications of integrality

The construction of the operation digraph forces certain relationships between the vehicle flow and the commodity flow variables used in NCSP1. Any arc leaving a pickup child vertex should transport at least one item of the associated item type. Likewise, any arc entering a delivery child vertex should transport at least one item of the associated item type.

Proposition 6: The following inequalities are valid for NCSP1, where $(i, j) \in \bar{A}$ and $k \in K$:

$$y_{ijk} \geq x_{ij} \quad (i \in P, a_{\pi(i)} = k \text{ or } j \in D, b_{\pi(j)} = k). \tag{22}$$

Inequalities (22) imply that an arc leaving a pickup vertex or entering a delivery vertex must be transporting at least one item of the associated item type. Based on the same reasoning, we now derive upper bounds for the commodity flow variables. Let $\beta_k = \sum_{i \in P} \alpha_{ik}$ denote the total amount of supply of items of type k, and let θ_{ijk} denote the upper bound on y_{ijk}:

$$\theta_{ijk} = \begin{cases} \min\{\beta_k, \lfloor (Q - q_{a_{\pi(j)}})/q_k \rfloor\} & i,j \in P, a_{\pi(i)} = k, \\ \min\{\beta_k, \lfloor (Q - q_{a_{\pi(i)}} - q_{a_{\pi(j)}})/q_k \rfloor\} & i,j \in P, a_{\pi(i)} \neq k, \\ \min\{\beta_k, \lfloor Q/q_k \rfloor\} & i \in P, j \in D, a_{\pi(i)} = k, b_{\pi(j)} = k, \\ \min\{\beta_k, \lfloor (Q - q_{b_{\pi(j)}})/q_k \rfloor\} & i \in P, j \in D, a_{\pi(i)} = k, b_{\pi(j)} \neq k, \\ \min\{\beta_k, \lfloor (Q - q_{a_{\pi(i)}})/q_k \rfloor\} & i \in P, j \in D, a_{\pi(i)} \neq k, b_{\pi(j)} = k, \\ \min\{\beta_k, \lfloor (Q - q_{a_{\pi(i)}} - q_{b_{\pi(j)}})/q_k \rfloor\} & i \in P, j \in D, a_{\pi(i)} \neq k, b_{\pi(j)} \neq k, \\ & \quad a_{\pi(i)} \neq b_{\pi(j)}, \\ \min\{\beta_k, \lfloor (Q - q_{a_{\pi(i)}})/q_k \rfloor\} & i \in P, j \in D, a_{\pi(i)} \neq k, b_{\pi(j)} \neq k, \\ & \quad a_{\pi(i)} = b_{\pi(j)}, \\ \min\{\beta_k, \lfloor (Q - q_{b_{\pi(i)}})/q_k \rfloor, \lfloor (Q - q_{a_{\pi(j)}})/q_k \rfloor\} & i \in D, j \in P, \\ \min\{\beta_k, \lfloor (Q - q_{b_{\pi(i)}})/q_k \rfloor\} & i,j \in D, b_{\pi(j)} = k, \\ \min\{\beta_k, \lfloor (Q - q_{b_{\pi(i)}} - q_{b_{\pi(j)}})/q_k \rfloor\} & i,j \in D, b_{\pi(j)} \neq k, \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 7: The following inequalities are valid for NCSP1.

$$y_{ijk} \leq \theta_{ijk} x_{ij} \quad ((i,j) \in \bar{A}, k \in K). \tag{23}$$

5.2 Benders inequalities

We first adapt some valid inequalities from the study of Hernández-Pérez and Salazar-González (2008). The authors have constructed three formulations for the m-PDTSP: a directed formulation with path variables, another with flow variables, and a third one in which the flow variables are eliminated by Benders decomposition. As in the latter formulation, we fix the x variables as x^* and take the dual of the subproblem consisting of maximizing 0, subject to (7), (8), and (9). The resulting dual subproblem is:

(DSP)

$$\text{minimize} \sum_{i \in \bar{V}, k \in K} \alpha_{ik} u_{ik} + \sum_{(i,j) \in \bar{A}} Q_{ij} x^*_{ij} w_{ij} \tag{24}$$

subject to

$$u_{ik} - u_{jk} + q_k w_{ij} \geq 0 \quad ((i,j) \in \bar{A}, k \in K : \theta_{ijk} > 0) \tag{25}$$
$$w_{ij} \geq 0 \quad ((i,j) \in \bar{A}). \tag{26}$$

Note that $\theta_{ijk} = 0$ implies $y_{ijk} = 0$. Consequently, y_{ijk} can be removed from the primal problem and the corresponding constraint can be removed from the dual problem. Hence we have reduced the constraint set (25) accordingly. If the objective function value (24) is less than 0, then the primal multicommodity flow problem is infeasible. Hence the following inequality is valid for NCSP2:

$$\sum_{i \in \bar{V}, k \in K} \alpha_{ik} u^*_{ik} + \sum_{(i,j) \in \bar{A}} Q_{ij} w^*_{ij} x_{ij} \geq 0. \tag{27}$$

Reorganizing the terms and using nonnegativity of the variables and parameters involved, we obtain the following *rounded Benders inequalities*:

$$\sum_{(i,j) \in \bar{A}} \lceil Q_{ij} w^*_{ij} \rceil x_{ij} \geq \lceil - \sum_{i \in \bar{V}, k \in K} \alpha_{ik} u^*_{ik} \rceil. \tag{28}$$

Inequalities (27) and (28) cannot be adapted to NCSP3. The reason is that the undirected variables in NCSP3 do not contain enough information to be used in DSP.

5.3 Clique cluster inequalities

Clique cluster inequalities are an adaptation of the *rank inequalities* for the *Stable Set Problem* (SSP). Hernández-Pérez and Salazar-González (2007) have adapted the clique inequalities to the 1-PDTSP by considering subsets of vertices $W_1, ..., W_m$ satisfying the following conditions: 1) all subsets intersect at a single vertex v ($W_k \cap W_l = \{v\}, \quad 1 \leq k < l \leq m$); 2) the capacity requirement of any subset does not exceed the vehicle capacity; 3) the capacity requirement of the union of any two subsets exceeds the vehicle capacity. By construction of the customer subsets, only customers in one subset W_k can be visited consecutively, implying $z(\delta(W_k)) = 2$. The remaining subsets W_l must satisfy $z(\delta(W_l)) \geq 4$. Using the same undirected edge variables in NCSP1, and summing up equality and the following inequalities the authors obtain:

$$\sum_{k=1}^{m} z(\delta(W_k)) \geq 4m - 2. \tag{29}$$

We first adapt the clique cluster inequalities for NCSP2. We need certain extra conditions to appropriately handle the transshipment vertices. We restate the second condition as: 2) for every customer subset W_k, $r^+(W_k) = r^-(W_k) = r^+(W_k \setminus \{v\}) = r^-(W_k \setminus \{v\}) = 1$. This ensures that every subset must be visited, which may not be the case in the presence of transshipment vertices. We also update the third condition as: 3) for the union of any two subsets W_k and W_l, $r^+(W_k \cap W_l) > 1$ and $r^-(W_k \cap W_l) > 1$. This modification handles the possible case of exceeding the vehicle capacity in the negative direction.

Proposition 8: The following inequalities are valid for NCSP2:

$$\sum_{k=1}^{m} x(\delta^+(W_k)) \geq 2m - 1, \tag{30}$$

$$\sum_{k=1}^{m} x(\delta^-(W_k)) \geq 2m - 1. \tag{31}$$

Proof: Following the same reasoning as above, no two sets of customers W_k and W_l can be visited consecutively, implying that the outflow and inflow of at least $m - 1$ subsets are two, and at most one subset is one. Hence the total inflow and outflow are at least $2(m - 1) + 1 = 2m - 1$. □

The adaptation process is more complicated for NCSP3. Let m_1 denote the number of subsets that include at least one element of P_T but not its sibling in D_T, and let $m_2 = m - m_1$.

Proposition 9: The following inequalities are valid for NCSP3:

$$\sum_{k=1}^{m} z(\delta(W_k)) \geq \max\{3m - 1, 3m_1 + 4m_2 - 2\} \tag{32}$$

$$\sum_{k=1}^{m} z(\delta(W_k) \setminus \bar{E}_T) + 2\, z(\delta(W_k) \cap \bar{E}_T) \geq 4m - 2. \tag{33}$$

Proof: Note that for 1-PDTSP, when a subset W_k cannot be visited in its entirety, the condition $z(\delta(W_k)) \geq 4$ holds. This condition may fail for NCSP3 if a pickup

child of a transshipment vertex is in a subset W_k and its sibling is not, and vice versa. In this case $z(\delta(W_k))$ may be equal to 3 since the sibling vertices may be connected to each other and the rest of the vertices in the subset may be visited consecutively. For the case of certain subsets violating this condition, (32) is valid. A second valid inequality is obtained by multiplying by two the variables corresponding to edges in E_T, which will decrease down to two the degree requirement of the associated customer subset in case it is used. □

Proposition 10: Inequalities (32) dominate (33).

Proof: We can rewrite (33) as

$$\sum_{k=1}^{m} z(\delta(W_k)) \geq 4m - 2 - \sum_{k=1}^{m} z(\delta(W_k) \cap \bar{E}_T). \tag{34}$$

Note that the left-hand sides of (32) and (34) are equal. Hence we only need to show that

$$\max\{3m - 1, 3m_1 + 4m_2 - 2\} \geq 4m - 2 - \sum_{k=1}^{m} z(\delta(W_k) \cap \bar{E}_T). \tag{35}$$

Case 1) $3m - 1 > 3m_1 + 4m_2 - 2$.

Using the fact $m_1 + m_2 = m$, the above condition simplifies to $m_2 < 1$, or equivalently, to $m_2 = 0$ and $m = m_1$. Substituting these values into (35), we obtain:

$$\sum_{k=1}^{m} z(\delta(W_k) \cap \bar{E}_T) \geq m_1 - 1. \tag{36}$$

Recall that m_1 is the number of subsets that are connected to at least one edge in E_T. Hence inequality (36) holds by definition of m_1.

Case 2) $3m - 1 \leq 3m_1 + 4m_2 - 2$

Using the fact $m_1 + m_2 = m$ and reorganizing the terms, we find

$$\sum_{k=1}^{m} z(\delta(W_k) \cap \bar{E}_T) \geq m_1. \tag{37}$$

By the same reasoning in case 1, the inequality (37) holds by the definition of m_1. □

5.4 Multistar inequalities

The following *multistar inequalities* are known to be valid for the undirected formulation of the 1-PDTSP (Hernández-Pérez and Salazar-González 2004). The *positive multistar inequality* is

$$z(\delta(N)) \geq \frac{2}{Q} \sum_{i \in N} (q_i + \sum_{j \in S} q_j z_{ij}) \tag{38}$$

for $N \subset V$ and $S = \{j \in V \setminus N : q_j > 0\}$. The *negative multistar inequality* is

$$z(\delta(N)) \geq \frac{2}{Q} \sum_{i \in N} (-q_i - \sum_{j \in S} q_j z_{ij}) \tag{39}$$

for $N \subset V$ and $S = \{j \in V \setminus N : q_j < 0\}$. The main idea behind these valid inequalities is that either the vehicle capacity should suffice to visit all vertices in the set N and two more in the set S, or the vehicle cannot visit all these vertices consecutively. In the case of the NCSP, the transshipment vertices can cause a degree of 3 instead of 4 when the capacity is exceeded. Hence the adapted versions of these inequalities take this fact into account.

Proposition 11: The following inequalities are valid for both NCSP1 and NCSP2:

$$x(\delta^+(N)) \geq \frac{1}{Q} \sum_{i \in N} (d(i) + \sum_{(i,j) \in A(N:P \setminus N)} d(j) x_{ij}) \tag{40}$$

for $N \subset P \cup D$, and

$$x(\delta^-(N)) \geq \frac{1}{Q} \sum_{i \in N} (-d(i) - \sum_{(j,i) \in A(N:D \setminus N)} d(j) x_{ji}) \tag{41}$$

for $N \subset P \cup D$.

For NCSP3, the right-hand side value is computed to account for two edges in $\delta(N)$ every time the vehicle capacity is exceeded. This computation is hard to modify for the transshipment vertices, since they can act like normal vertices. Instead, we proceed as for the clique cluster inequalities and multiply the connections between sibling transshipment vertices by 2. The positive multistar inequality becomes

$$z(\delta(N) \setminus \bar{E}_T) + 2\, z(\delta(N) \cap \bar{E}_T) \geq \frac{2}{Q} \sum_{i \in N} (d(i) + \sum_{(i,j) \in \bar{E}: i \in N, j \in P \setminus N} d(j) z_{ij}) \qquad (42)$$

for $N \subset P \cup D$. Similarly, the negative multistar inequality becomes

$$z(\delta(N) \setminus \bar{E}_T) + 2\, z(\delta(N) \cap \bar{E}_T) \geq \frac{2}{Q} \sum_{i \in N} (-d(i) - \sum_{(i,j) \in \bar{E}: i \in N, j \in D \setminus N} d(j) z_{ij}) \qquad (43)$$

for $N \subset P \cup D$.

6 Branch-and-Cut Algorithm

In this section we present the separation algorithms for the valid inequalities, our iterated local search heuristic, and a unified branch-and-cut algorithm for our three formulations.

6.1 Separation algorithms

To separate the rounded capacity constraints, the multistar inequalities, and the clique cluster inequalities, we have used the separation algorithms described in Hernández-Pérez and Salazar-González (2007). For the Benders inequalities and the rounded Benders inequalities, we have solved the DSP, as described in Hernández-Pérez and Salazar-González (2008).

6.2 Heuristic

We have developed an iterated local search heuristic for the NCSP. Its first step uses a nearest neighbor heuristic which we now describe.

Nearest neighbor for NCSP

This algorithm operates on the operation digraph.

Step 1 (Initialization). Connect all transshipment vertices to their siblings, and mark them as "visited". Mark the rest of the vertices as "unvisited". Set the on-board inventory of every item type to be zero. Set the depot to be the last visited vertex.

Step 2 (Vertex selection). Select the closest vertex to the last visited vertex from unmarked elements of $P \cup D$, visiting which will not decrease the on-board inventory of any item type to less than zero, or increase the total weight of the carried items to more than vehicle capacity.

Step 3 (Vertex insertion). If the selected vertex is an element of P, add the associated supply item to on-board inventory and increase the total weight accordingly. Else if the selected vertex is an element of D, remove the associated demand item from the on-board inventory and decrease the total weight accordingly. Set the selected vertex as the last visited vertex. If there is at least one unvisited vertex in $P \cup D$, go to Step 2.

Step 3 (Termination). Connect the last visited vertex to the second copy of the depot.

Iterated local search

Our iterated local search heuristic works with three parameters: κ is the number of iterations since the last update of the best solution value, λ is the maximum number of iterations without updating the best solution, and γ is a measure of perturbation of the solution at hand.

Step 1 (Initialization). Construct a feasible solution by using the nearest neighbor algorithm for the NCSP. Set $\kappa = 1$ and γ to be equal to one quarter of the number of vertices in the initial solution.

Step 2 (Termination criterion). If $\kappa < \lambda$, go to Step 3. Else, stop.

Step 3 (Perturbation). Initialize a perturbed solution by copying the best known solution. For γ times, randomly select two vertices in the perturbed solution and swap them.

Step 4 (Local search). Apply the best move selected among 2-exchange and removal-reinsertion moves on the perturbed solution, penalizing the objective function value by the value of the best known solution for every arc where the on-board amount of an item type decreases below 0, the vehicle capacity is exceeded, or the arc does not exist. If the final perturbed solution has a lower cost, update the best known solution. Increment κ by 1 and go to Step 2.

In our experimentation we have used $\lambda = 10000$. Note that since the initial solution supplied by the nearest neighbor algorithm does not contain any transshipment vertices, neither does the final solution returned by the iterated local search algorithm. Our computational experiments, which we present in Section 7, showed that the transshipment vertices are rarely used. Hence this restriction does not have a major impact on the performance of our heuristic.

6.3 Step-by-step description of the algorithm

We now describe our unified branch-and-cut algorithm capable of handling all formulations.

Step 1 (Upper bound). Compute an upper bound on the optimal solution of the problem, using the iterated local search heuristic.

Step 2 (Root node). Construct the corresponding formulation without the sets of constraints that have an exponential size. Insert this subproblem in a list.

Step 3 (Node selection). If the list is empty, stop. Else select and remove a subproblem from the list.

Step 4 (Subproblem solution). Solve the subproblem. If the objective function value is greater than the best upper bound, go to Step 3.

Step 5 (Constraint generation). Identify violated members of the associated constraints and related valid inequalities and add them to the subproblem. If at least one constraint is generated, go to Step 4.

Step 6 (Integrality check). If the solution is integer, update the best known upper bound, and go to Step 3.

Step 7 (Branching). Construct two subproblems by branching on a binary fractional variable. Add the subproblems to the list and go to Step 3.

The subproblems are solved by CPLEX, where the *strong branching* option was used in Step 7.

7 Computational Results

To test the formulations and algorithms presented in the previous sections, we have conducted computational experiments on 64 instances adapted from TSPLIB instances. To ensure the reproducibility of our results, we have adapted the data by using the coordinates of the vertices as our random number stream. We have used

the y coordinates to minimize bias since the x coordinates have been sorted in ascending order for certain instances. We use different modular values μ_1, μ_2 and μ_3 to obtain three independent random number streams out of one. We now present the algorithm we use to convert the instances:

Step 1 (Initialization). Set the first vertex in the data file to be the depot. Construct two vectors, (a) and (b) of size $|V|$ to denote the supply and demand at the vertices $i \in V$. Construct two more vectors (p) and (q) of size $|V|$. These vectors will be used to randomize the order of the supply and demand vectors.

Step 2 (Supply / demand assignment). Set $a_i = i + 1$, $i \in \{0, ..., |K| - 1\}$, $a_i = \lfloor y_i \rfloor \mod |K| + 1$ for $i \in \{|K|, ..., |V| - 1\}$ and $b_i = a_i$ for $i \in V$. With this assignment, we ensure that every type of item exists in the data and the supply of every type of item equals its demand. Item type 0 denotes the case of no demand or no supply.

Step 3 (Randomization). Set $p_i = \lfloor y_i \rfloor \mod \mu_1$ and $q_i = \lfloor y_i \rfloor \mod \mu_2$ for $i \in V$. Sort the vector (p), applying the same swaps on the vector (a). Similarly, sort the vector (q), applying the same swaps on the vector (b). This step ensures a random distribution of demand and supply.

Step 4 (Correction). If $a_i = b_i = 0$ for some $i \in V \setminus \{0\}$, set $a_i = b_i = (\lfloor y_i \rfloor \mod |K - 1|) + 1$. This step converts vertices with no supply and no demand into transshipment vertices. Note that a_0 and b_0 are allowed to have the value of 0.

Step 5 (Weight assignment). Set the weight of the item type $k \in K$ as $q_k = (\lfloor y_k \rfloor \mod \mu_3) + 1$.

In our experimentation we have used $\mu_1 = 11$, $\mu_2 = 13$, and $\mu_3 = 5$. The distances between the vertices were computed using the Euclidean distance formula and rounded up to the closest integer.

We have applied the algorithm given above to four instances from the TSPLIB: `brd14051`, `fnl4461`, `nrw1379`, and `pr1002`. We have used $|K| = 2, 4, 6, 8$ to analyze the effect of the number of item types on the computational performance. We have determined the vehicle capacity based on a scaling parameter $\psi \in [0, 1]$ as $Q = \max_{k \in K} q_k + \psi(\sum_{i \in P} q_{a_{\pi(i)}} - \max_{k \in K} q_k)$. This setting ensures that the vehicle capacity is sufficient to transport every item type when $\psi = 0$, and to transport all items at the same time when $\psi = 1$.

We have implemented NCSP1, NCSP2, NCSP3, and the iterated local search heuristic using C++ and CPLEX 10.1, and we have conducted computational experiments on a Linux workstation with a 64-bit AMS Opteron 250 CPU running

at 2.4GHz, and 16 GB of RAM. To compare the performance of our formulations we have used instances with $|V| = 15$, which roughly corresponds to $|\bar{V}| = 30$. We have observed that increasing ψ to more than 0.3 does not effect the results, so we have used the values of $\psi = 0, 0.1, 0.2, 0.3$. The results are presented in Tables 1 to 4, with objective function values based on the modified cost coefficients for $\bar{G}$.

The results are somewhat in contrast with our expectations. Our first observation is that NCSP3 behaves considerably worse than NCSP2. A deeper analysis of the formulations reveals three reasons. In the operation digraph, there is only one arc between the sibling vertices of non-transshipment vertices, and that arc is from the delivery child vertex to the pickup child vertex. Remember that this choice eliminates the possibility of a violation of vehicle capacity between the pickup and delivery operations at the same parent vertex, as well as the symmetry of solutions that deliver first and pickup second, and that those pickup first and deliver second. For the case of NCSP3, possible capacity violations between the sibling vertices cannot be ruled out by the direction of arcs, so they should be identified and cut off by valid inequalities. The symmetry mentioned above cannot be avoided in the case of the operation graph. Given a solution consisting of a single visit to q vertices involving both supply and demand, and both operations can feasibly be performed in any order, one can construct $2^q - 1$ equivalent solutions by changing the order and pickup and delivery for these q vertices. Hence converting directed vehicle flow variables into undirected vehicle flow varibles increases the symmetry rather than decreasing it. Finally, Benders inequalities, which are observed to be quite effective for harder instances, cannot be applied to NCSP3 because of the lack of directional information. We conclude that direction is a valuable information for the NCSP, contrary to what happens in the 1-PDTSP.

Our second observation is that NCSP1 performs better than NCSP2, despite the fact that it involves more variables and lacks the rounded capacity constraints. Not only does NCSP1 succesfully solve four instances that NCSP2 cannot, but it also performs better in terms of the initial gap, the final gap and the CPU time. The basis for the better performance seems to be the valid inequalities based on the implications of integrality presented in Section 5. Including these inequalities in the Benders decomposition increases the subproblem solution time considerably for NCSP2. When combined with the separation effort for the rest of the valid inequalities, a high separation time is required to separate the Benders inequalities, which NCSP1 handles with routine simplex iterations. Hence the same valid inequalities cannot be used to boost the performance of NCSP2. Knowing that the rounded capacity constraints dominate the connectivity constraints, we have replaced them

with the rounded capacity constraints of NCSP2 to obtain NCSP1'. This final formulation has the best performance, successfully solving the 64 instances within an hour of computing time, within an average of just over three minutes.

We next proceeded to test the performance of NCSP1' on larger instances. Tables 5 and 6 show the results of our experiments for $|V| = 20$, which roughly corresponds to $|\bar{V}| = 40$. For this set of instances we were able to solve 57 out of 64 instances. The final gap is no more than 3% for five of the seven instances that could not be solved. The average CPU time requirement is 2222 seconds. The computational reach of NCSP1' extends to $|V| = 30$ for $|K| = 2$ and 4. Table 7 shows that we were able to solve 31 instances out of 32 in about one hour of computing time. The average CPU time requirement is about 700 seconds for these instances.

CPLEX usually fails to find good solutions in the earlier stages of the branch-and-cut algorithm, hence the results supplied by the iterated local search helped us extend our computational reach. The CPU time requirement for the heuristic was never more than two minutes. It finds the optimum solution for 26 of the 64 instances with $|V| = 15$, for 40 instances out of 64 for $|V| = 20$, and for 20 instances out of 32 for $|V| = 30$. For NCSP1', it provides an average initial gap of 1.05%, 2.70%, and 1.92% for $|V| = 15, 20$, and 30, respectively.

8 Conclusions

In this paper we have analyzed the NCSP, proposed three integer programming formulations based on a graph transformation, presented several families of valid inequalities, and developed a unified branch-and-cut algorithm. We have also implemented a basic iterated local search algorithm to yield an initial feasible solution. In contrast with what is observed on related problems, flow based formulations have a better computational performance than the directed and undirected vehicle flow formulations. For six to eight item types, we were able to solve instances with $|V| = 20$. For two to four item types, our computational reach was observed to be up to $|V| = 30$.

Acknowledgments: This work was partially funded by the Canadian Natural Sciences and Engineering Research Council under grants 227837-04 and 39682-05. This support is gratefully acknowledged.

References

Anily, S., M. Gendreau, G. Laporte. 1999. The swapping problem on a line. *SIAM Journal on Computing* **29** 327–335.

Anily, S., M. Gendreau, G. Laporte. 2008. The preemptive swapping problem on a tree. Tech. rep., GERAD, Montreal, Canada.

Anily, S., R. Hassin. 1992. The swapping problem. *Networks* **22** 419–433.

Atallah, M. J., S. R. Kosaraju. 1988. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing* **17** 849–869.

Ball, M. O., M. J. Magazine. 1988. Approximating capacitated routing and delivery problems. *Operations Research* **36** 192–201.

Bordenave, C., M. Gendreau, G. Laporte. 2008a. Heuristics for the mixed swapping problem. Submitted to *Computers & Operations Research*.

Bordenave, C., M. Gendreau, G. Laporte. 2008b. The non-preemptive swapping problem. Submitted to *Naval Research Logistics*.

Bordenave, C., M. Gendreau, G. Laporte. 2008c. The preemptive swapping problem. Submitted to *Mathematical Programming*.

Chalasani, P., R. Motwani. 1993. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing* **15** 29–60.

Escudero, L. F. 1988. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research* **37** 236–249.

Frederickson, G. N., D. J. Guan. 1992. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing* **21** 1130–1152.

Frederickson, G. N., D. J. Guan. 1993. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms* **15** 29–60.

Frederickson, G. N., M. S. Hecht, C. E. Kim. 1978. Approximation algorithms for some routing problems. *SIAM Journal on Computing* **7** 178–193.

Hernández-Pérez, H., J.-J. Salazar-González. 2004. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics* **145** 126–139.

Hernández-Pérez, H., J.-J. Salazar-González. 2007. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks* **50** 258–272.

Hernández-Pérez, H., J.-J. Salazar-González. 2008. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *Unpublished manuscript* .

Sherali, H.D. 1982. Equivalent weights for lexicographic multi-objective programs: Characterizations and computations. *European Journal of Operational Research* **11** 367–379.

					NCSP1				NCSP2			
TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B&C nodes	CPU time (sec)	Initial gap	Final gap	B&C nodes	CPU time (sec)	Initial gap	Final gap
brd14051	2	0	88362.0	88362.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	75819.0	75819.0	1	0.3	0.00%	0.00%	0	0.1	0.00%	0.00%
	2	0.2	75498.0	75498.0	1	0.8	0.00%	0.00%	0	0.1	0.00%	0.00%
	2	0.3	75498.0	75498.0	1	0.8	0.00%	0.00%	0	0.1	0.00%	0.00%
	4	0	154190.0	154093.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	4	0.1	134415.0	134029.0	2	1.1	0.73%	0.00%	0	0.2	0.00%	0.00%
	4	0.2	132753.0	127438.0	1	1.4	0.00%	0.00%	0	0.6	0.00%	0.00%
	4	0.3	127438.0	127438.0	2	3.9	0.03%	0.00%	0	0.8	0.00%	0.00%
	6	0	161329.0	157585.0	2	0.7	0.04%	0.00%	102	29.9	4.31%	0.00%
	6	0.1	148949.0	138257.0	32	38.2	1.19%	0.00%	531	4186.2	3.72%	0.00%
	6	0.2	133395.0	128882.0	79	157.3	0.79%	0.00%	4	12.8	0.57%	0.00%
	6	0.3	128882.0	128882.0	134	226.3	0.91%	0.00%	8	13.7	0.54%	0.00%
	8	0	201235.0	192656.0	1	0.1	0.00%	0.00%	1	0.3	0.00%	0.00%
	8	0.1	138004.0	133651.0	60	62.8	0.12%	0.00%	42	50.6	0.20%	0.00%
	8	0.2	147509.0	132498.0	6126	12999.2	2.92%	0.00%	395	394.2	2.72%	0.00%
	8	0.3	133491.0	130322.0	990	2170.6	1.64%	0.00%	19	32.5	1.14%	0.00%
fnl4461	2	0	55912.0	55912.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	53288.0	53288.0	1	0.3	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.2	53256.0	53256.0	1	1.5	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.3	53256.0	53256.0	1	1.5	0.00%	0.00%	0	0.2	0.00%	0.00%
	4	0	57644.0	57644.0	1	0.1	0.00%	0.00%	0	0.2	0.00%	0.00%
	4	0.1	56812.0	56111.0	1	0.2	0.00%	0.00%	7	2.9	0.81%	0.00%
	4	0.2	55182.0	55087.0	19	10.6	0.57%	0.00%	15	2.7	0.62%	0.00%
	4	0.3	55149.0	55022.0	35	16.0	0.96%	0.00%	12	1.7	0.29%	0.00%
	6	0	100880.0	97586.0	5	1.5	2.46%	0.00%	46	12.1	4.60%	0.00%
	6	0.1	80177.0	80177.0	9	3.6	0.70%	0.00%	56	45.6	16.93%	0.00%
	6	0.2	59086.0	56464.0	349	320.5	2.61%	0.00%	164	76.7	3.10%	0.00%
	6	0.3	59021.0	55951.0	447	887.1	2.41%	0.00%	94	33.4	2.26%	0.00%
	8	0	103315.0	102384.0	8058	6582.6	22.88%	0.00%	1085	14400.4	27.82%	10.02%
	8	0.1	61906.0	58803.0	221	322.6	1.16%	0.00%	48	52.9	1.50%	0.00%
	8	0.2	59120.0	58641.0	1136	1754.2	4.82%	0.00%	3320	3875.4	5.72%	0.00%
	8	0.3	56528.0	56049.0	301	601.6	1.89%	0.00%	97	92.0	1.67%	0.00%
Average					562.7	817.7	1.53%	0.00%	188.9	728.7	2.45%	0.31%

Table 1: Computational results for NCSP1 and NCSP2 for `brd14051` and `fnl4461` with $|V| = 15$.

					NCSP1				NCSP2			
TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B&C nodes	CPU time (sec)	Initial gap	Final gap	B&C nodes	CPU time (sec)	Initial gap	Final gap
nrw1379	2	0	71592.0	71592.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	63272.0	63272.0	1	0.4	0.00%	0.00%	0	0.1	0.00%	0.00%
	2	0.2	63272.0	63272.0	1	0.8	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.3	63272.0	63272.0	1	0.7	0.00%	0.00%	0	0.2	0.00%	0.00%
	4	0	169035.0	169035.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	4	0.1	117804.0	117677.0	13	3.3	19.18%	0.00%	2157	14400.2	20.67%	6.45%
	4	0.2	77324.0	77230.0	29	19.2	1.17%	0.00%	14	3.3	0.14%	0.00%
	4	0.3	77324.0	77133.0	150	92.2	1.88%	0.00%	0	1.0	0.00%	0.00%
	6	0	156113.0	155789.0	1	0.4	0.00%	0.00%	749	414.2	34.18%	0.00%
	6	0.1	93137.0	87664.0	6108	7698.5	9.32%	0.00%	567	14497.5	12.48%	7.60%
	6	0.2	77935.0	77935.0	59	143.1	1.92%	0.00%	10	21.4	1.54%	0.00%
	6	0.3	77935.0	77935.0	102	184.9	2.10%	0.00%	12	18.3	1.61%	0.00%
	8	0	154065.0	153679.0	1	0.1	0.00%	0.00%	9	1.2	2.24%	0.00%
	8	0.1	86225.0	85808.0	84	96.3	3.85%	0.00%	451	14427.5	6.69%	3.58%
	8	0.2	81296.0	79887.0	118	386.2	2.91%	0.00%	135	188.4	3.32%	0.00%
	8	0.3	84656.0	79887.0	196	843.6	3.50%	0.00%	76	133.6	3.48%	0.00%
pr1002	2	0	383309.0	383309.0	1	0.0	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	355947.0	344651.0	1	0.3	0.00%	0.00%	0	0.1	0.00%	0.00%
	2	0.2	341131.0	341131.0	1	0.5	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.3	355403.0	341131.0	1	0.6	0.00%	0.00%	0	0.2	0.00%	0.00%
	4	0	314726.0	314726.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	4	0.1	293478.0	293478.0	1	0.2	0.00%	0.00%	0	0.4	0.00%	0.00%
	4	0.2	293478.0	293478.0	1	1.0	0.00%	0.00%	0	0.8	0.00%	0.00%
	4	0.3	293478.0	293478.0	1	0.9	0.00%	0.00%	0	0.5	0.00%	0.00%
	6	0	626608.0	603216.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	6	0.1	437359.0	428593.0	5	4.0	1.12%	0.00%	28	140.7	4.81%	0.00%
	6	0.2	418160.0	390448.0	47	109.8	2.30%	0.00%	37	45.3	3.52%	0.00%
	6	0.3	387281.0	387281.0	112	137.4	5.20%	0.00%	51	41.6	2.76%	0.00%
	8	0	504752.0	486958.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	8	0.1	460433.0	420079.0	5	10.1	0.74%	0.00%	137	979.2	5.67%	0.00%
	8	0.2	430703.0	391056.0	39	109.5	3.05%	0.00%	47	120.4	4.78%	0.00%
	8	0.3	392976.0	384529.0	48	150.9	3.43%	0.00%	11	48.4	3.03%	0.00%
Average					222.3	312.3	1.93%	0.00%	140.3	1421.4	3.47%	0.55%

Table 2: Computational results for NCSP1 and NCSP2 for `nrw1379` and `pr1002` with $|V| = 15$.

					NCSP3				NCSP1'			
TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B&C nodes	CPU time (sec)	Initial gap	Final gap	B&C nodes	CPU time (sec)	Initial gap	Final gap
brd14051	2	0	88362.0	88362.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	75819.0	75819.0	3	0.1	0.04%	0.0%	0	0.1	0.00%	0.00%
	2	0.2	75498.0	75498.0	1	0.1	0.00%	0.00%	0	0.5	0.00%	0.00%
	2	0.3	75498.0	75498.0	1	0.1	0.00%	0.00%	0	0.5	0.00%	0.00%
	4	0	154190.0	154093.0	26455	5883.7	2.95%	0.00%	0	0.0	0.00%	0.00%
	4	0.1	134415.0	134029.0	303	5.1	2.45%	0.00%	2	0.7	0.05%	0.00%
	4	0.2	132753.0	127438.0	7	0.2	0.06%	0.00%	0	1.5	0.00%	0.00%
	4	0.3	127438.0	127438.0	8	0.1	0.08%	0.00%	1	2.4	0.00%	0.00%
	6	0	161329.0	157585.0	15628	14400.3	18.85%	16.16%	1	0.3	0.00%	0.00%
	6	0.1	148949.0	138257.0	19111	14400.2	8.45%	10.25%	18	18.2	0.96%	0.00%
	6	0.2	133395.0	128882.0	343	10.4	1.16%	0.00%	4	15.2	0.08%	0.00%
	6	0.3	128882.0	128882.0	1678	45.1	1.16%	0.00%	17	47.3	0.14%	0.00%
	8	0	201235.0	192656.0	9786	14400.7	46.81%	42.23%	0	0.0	0.00%	0.00%
	8	0.1	138004.0	133651.0	29942	6768.8	2.66%	0.00%	7	7.9	0.09%	0.00%
	8	0.2	147509.0	132498.0	62705	14400.0	4.00%	1.02%	621	2365.7	2.31%	0.00%
	8	0.3	133491.0	130322.0	17772	1180.7	2.29%	0.00%	21	80.0	1.09%	0.00%
fnl4461	2	0	55912.0	55912.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	53288.0	53288.0	1	0.1	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.2	53256.0	53256.0	1	0.3	0.00%	0.00%	0	0.9	0.00%	0.00%
	2	0.3	53256.0	53256.0	1	0.3	0.00%	0.00%	0	0.7	0.00%	0.00%
	4	0	57644.0	57644.0	163	2.8	4.59%	0.00%	0	0.1	0.00%	0.00%
	4	0.1	56812.0	56111.0	328	11.9	3.40%	0.00%	0	0.2	0.00%	0.00%
	4	0.2	55182.0	55087.0	133	3.0	2.26%	0.00%	11	6.9	0.37%	0.00%
	4	0.3	55149.0	55022.0	65	0.8	2.14%	0.00%	10	7.1	0.61%	0.00%
	6	0	100880.0	97586.0	10007	14400.1	76.47%	61.65%	0	0.4	0.00%	0.00%
	6	0.1	80177.0	80177.0	14296	14400.0	48.30%	41.74%	2	2.0	0.12%	0.00%
	6	0.2	59086.0	56464.0	5376	688.3	4.60%	0.00%	63	52.6	2.00%	0.00%
	6	0.3	59021.0	55951.0	1185	60.6	3.68%	0.00%	79	165.4	2.22%	0.00%
	8	0	103315.0	102384.0	12323	14400.1	77.71%	68.70%	3795	3983.4	16.41%	0.00%
	8	0.1	61906.0	58803.0	43910	14400.1	6.33%	0.41%	31	27.2	0.59%	0.00%
	8	0.2	59120.0	58641.0	18271	14400.2	8.60%	4.23%	499	701.0	3.69%	0.00%
	8	0.3	56528.0	56049.0	2219	317.5	3.80%	0.00%	129	257.7	1.56%	0.00%
Average					9125.6	4518.2	10.40%	7.70%	166.0	242.1	1.01%	0.00%

Table 3: Computational results for NCSP3 and NCSP1' for `brd14051` and `fnl4461` with $|V| = 15$.

					NCSP3				NCSP1'			
TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B&C nodes	CPU time (sec)	Initial gap	Final gap	B&C nodes	CPU time (sec)	Initial gap	Final gap
nrw1379	2	0	71592.0	71592.0	1	0.1	0.00%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	63272.0	63272.0	1	0.1	0.00%	0.00%	0	0.2	0.00%	0.00%
	2	0.2	63272.0	63272.0	1	0.1	0.00%	0.00%	0	0.5	0.00%	0.00%
	2	0.3	63272.0	63272.0	1	0.1	0.00%	0.00%	0	0.5	0.00%	0.00%
	4	0	169035.0	169035.0	16294	14400.2	122.71%	36.44%	0	0.0	0.00%	0.00%
	4	0.1	117804.0	117677.0	16841	14400.1	55.48%	50.23%	7	2.8	4.88%	0.00%
	4	0.2	77324.0	77230.0	283	4.3	2.16%	0.00%	9	7.7	0.09%	0.00%
	4	0.3	77324.0	77133.0	244	3.3	2.03%	0.00%	4	8.9	0.00%	0.00%
	6	0	156113.0	155789.0	10347	14400.2	96.71%	88.62%	0	0.3	0.00%	0.00%
	6	0.1	93137.0	87664.0	19040	14400.1	15.86%	11.81%	1886	2654.7	7.00%	0.00%
	6	0.2	77935.0	77935.0	1018	34.3	3.01%	0.00%	13	42.5	1.43%	0.00%
	6	0.3	77935.0	77935.0	800	26.1	3.01%	0.00%	7	61.0	1.45%	0.00%
	8	0	154065.0	153679.0	9670	14400.2	103.13%	89.75%	2	0.2	0.00%	0.00%
	8	0.1	86225.0	85808.0	24692	14400.1	13.41%	8.45%	55	58.4	2.86%	0.00%
	8	0.2	81296.0	79887.0	48197	14400.0	5.59%	2.03%	90	302.5	2.71%	0.00%
	8	0.3	84656.0	79887.0	43798	8341.1	5.59%	0.00%	114	781.2	3.31%	0.00%
pr1002	2	0	383309.0	383309.0	23	0.2	5.49%	0.00%	0	0.0	0.00%	0.00%
	2	0.1	355947.0	344651.0	5	0.1	1.53%	0.00%	0	0.2	0.00%	0.00%
	2	0.2	341131.0	341131.0	7	0.1	3.71%	0.00%	0	0.4	0.00%	0.00%
	2	0.3	355403.0	341131.0	10	0.2	3.71%	0.00%	0	0.3	0.00%	0.00%
	4	0	314726.0	314726.0	9	0.1	8.99%	0.00%	0	0.0	0.00%	0.00%
	4	0.1	293478.0	293478.0	8	0.1	1.63%	0.00%	0	0.2	0.00%	0.00%
	4	0.2	293478.0	293478.0	8	0.1	1.63%	0.00%	0	0.7	0.00%	0.00%
	4	0.3	293478.0	293478.0	8	0.1	1.63%	0.00%	0	1.2	0.00%	0.00%
	6	0	626608.0	603216.0	34620	14400.1	57.34%	32.85%	0	0.0	0.00%	0.00%
	6	0.1	437359.0	428593.0	36825	14400.0	20.29%	3.31%	0	1.5	0.00%	0.00%
	6	0.2	418160.0	390448.0	1350	122.3	13.39%	0.00%	16	28.4	1.60%	0.00%
	6	0.3	387281.0	387281.0	760	45.1	12.47%	0.00%	61	146.2	3.65%	0.00%
	8	0	504752.0	486958.0	10488	1656.8	32.86%	0.00%	0	0.0	0.00%	0.00%
	8	0.1	460433.0	420079.0	22289	14400.2	22.00%	3.61%	1	2.3	0.00%	0.00%
	8	0.2	430703.0	391056.0	890	76.6	13.57%	0.00%	39	107.6	2.91%	0.00%
	8	0.3	392976.0	384529.0	377	16.0	11.67%	0.00%	15	39.7	2.99%	0.00%
Average					9340.7	4822.8	20.02%	10.22%	72.5	132.8	1.09%	0.00%

Table 4: Computational results for NCSP3 and NCSP1' for `nrw1379` and `pr1002` with $|V| = 15$.

TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B & C nodes	CPU time (sec)	Initial gap	Final gap
brd14051	2	0	171412.0	171412.0	1	0.1	0.00%	0.00%
	2	0.1	171412.0	171412.0	1	2.5	0.00%	0.00%
	2	0.2	171412.0	171412.0	1	3.9	0.00%	0.00%
	2	0.3	171412.0	171412.0	1	3.7	0.00%	0.00%
	4	0	419928.0	419928.0	1	0.0	0.00%	0.00%
	4	0.1	179731.0	175826.0	3	8.9	0.31%	0.00%
	4	0.2	178472.0	174441.0	50	252.1	1.13%	0.00%
	4	0.3	178472.0	174441.0	11	196.2	0.43%	0.00%
	6	0	252513.0	252513.0	1	0.6	0.00%	0.00%
	6	0.1	172589.0	172589.0	597	11962.9	3.03%	0.00%
	6	0.2	172589.0	172589.0	196	14400.5	3.37%	1.25%
	6	0.3	172589.0	172589.0	167	14400.3	3.40%	1.32%
	8	0	339962.0	336687.0	1	0.4	0.00%	0.00%
	8	0.1	190444.0	180028.0	763	4706.4	2.93%	0.00%
	8	0.2	178769.0	178726.0	305	14400.8	3.60%	0.81%
	8	0.3	178769.0	178726.0	235	14400.4	3.54%	1.24%
fnl4461	2	0	97413.0	97413.0	1	0.1	0.00%	0.00%
	2	0.1	75659.0	74904.0	1	1.3	0.00%	0.00%
	2	0.2	73558.0	72803.0	1	3.4	0.10%	0.00%
	2	0.3	72886.0	72131.0	2	4.6	0.44%	0.00%
	4	0	131644.0	131644.0	4	1.8	3.98%	0.00%
	4	0.1	79606.0	79606.0	28	26.9	2.94%	0.00%
	4	0.2	74230.0	74230.0	8	11.0	0.77%	0.00%
	4	0.3	74062.0	74062.0	5	11.8	0.32%	0.00%
	6	0	149750.0	149581.0	1	0.6	0.00%	0.00%
	6	0.1	100146.0	100146.0	5216	14400.1	5.28%	0.71%
	6	0.2	78224.0	78224.0	19	53.7	1.18%	0.00%
	6	0.3	77635.0	77635.0	68	374.9	1.93%	0.00%
	8	0	149919.0	145009.0	403	709.1	3.11%	0.00%
	8	0.1	115313.0	115313.0	2204	14400.3	32.60%	22.61%
	8	0.2	75665.0	75118.0	90	388.9	1.49%	0.00%
	8	0.3	74656.0	74613.0	214	1129.0	1.76%	0.00%
Average					330.9	3320.5	2.43%	0.87%

Table 5: Computational results for NCSP1' for `brd14051` and `fnl4461` with $|V| = 20$.

TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B & C nodes	CPU time (sec)	Initial gap	Final gap
nrw1379	2	0	97662.0	97622.0	1	0.1	0.00%	0.00%
	2	0.1	94429.0	94429.0	1	2.9	0.06%	0.00%
	2	0.2	94387.0	94387.0	1	3.8	0.00%	0.00%
	2	0.3	94387.0	94387.0	1	4.8	0.00%	0.00%
	4	0	195780.0	195738.0	1	0.1	0.00%	0.00%
	4	0.1	112791.0	107918.0	31	25.8	2.62%	0.00%
	4	0.2	106111.0	106111.0	27	86.5	3.77%	0.00%
	4	0.3	105817.0	105817.0	15	71.8	3.19%	0.00%
	6	0	254291.0	251601.0	1	0.4	0.00%	0.00%
	6	0.1	104266.0	104266.0	113	261.0	2.21%	0.00%
	6	0.2	101072.0	101072.0	11	155.9	1.98%	0.00%
	6	0.3	101072.0	101072.0	10	91.1	1.99%	0.00%
	8	0	250969.0	246266.0	12	6.1	0.67%	0.00%
	8	0.1	163358.0	163358.0	1906	14400.2	47.65%	36.68%
	8	0.2	110521.0	102709.0	39	592.6	1.42%	0.00%
	8	0.3	110521.0	102709.0	34	642.4	1.47%	0.00%
pr1002	2	0	768613.0	768613.0	1	0.1	0.00%	0.00%
	2	0.1	555798.0	555798.0	1	1.2	0.00%	0.00%
	2	0.2	555798.0	552271.0	2	6.1	1.21%	0.00%
	2	0.3	555798.0	552271.0	2	5.9	1.16%	0.00%
	4	0	594223.0	594223.0	1	0.1	0.00%	0.00%
	4	0.1	415723.0	415723.0	1	0.9	0.00%	0.00%
	4	0.2	407407.0	407407.0	1	4.4	0.00%	0.00%
	4	0.3	407407.0	407407.0	1	5.3	0.00%	0.00%
	6	0	1358340.0	1311298.0	1	0.1	0.00%	0.00%
	6	0.1	887816.0	861605.0	30	48.5	2.23%	0.00%
	6	0.2	591087.0	591087.0	120	559.5	4.12%	0.00%
	6	0.3	586130.0	586130.0	156	1586.8	5.28%	0.00%
	8	0	874584.0	840352.0	1	0.1	0.00%	0.00%
	8	0.1	689239.0	689239.0	43	86.0	2.77%	0.00%
	8	0.2	591589.0	584912.0	386	2926.7	3.89%	0.00%
	8	0.3	591589.0	591589.0	297	14400.6	7.57%	3.01%
Average					101.1	1124.3	2.98%	1.24%

Table 6: Computational results for NCSP1' for `nrw1379` and `pr1002` with $|V| = 20$.

TSPLIB instance	Item types	ψ	Initial incumbent	Best incumbent	B & C nodes	CPU time (sec)	Initial gap	Final gap
brd14051	2	0	484360.0	484360.0	1	0.2	0.00%	0.00%
	2	0.1	259983.0	259983.0	2	14.9	0.06%	0.00%
	2	0.2	259983.0	259983.0	1	22.0	0.02%	0.00%
	2	0.3	259983.0	259983.0	1	29.0	0.07%	0.00%
	4	0	487968.0	484927.0	1	0.2	0.00%	0.00%
	4	0.1	268424.0	268424.0	87	690.6	1.47%	0.00%
	4	0.2	268424.0	268424.0	49	915.7	1.83%	0.00%
	4	0.3	268424.0	268424.0	30	340.3	0.98%	0.00%
fnl4461	2	0	179760.0	179264.0	1	0.1	0.00%	0.00%
	2	0.1	138282.0	137972.0	52	79.0	3.77%	0.00%
	2	0.2	125200.0	125200.0	14	19.7	1.10%	0.00%
	2	0.3	125200.0	125200.0	11	13.3	1.18%	0.00%
	4	0	332221.0	328687.0	1	2.0	0.00%	0.00%
	4	0.1	209090.0	209090.0	2217	14400.2	31.44%	24.32%
	4	0.2	134255.0	134255.0	36	96.0	1.83%	0.00%
	4	0.3	133945.0	133945.0	24	99.6	1.76%	0.00%
nrw1379	2	0	395335.0	395335.0	1	0.0	0.00%	0.00%
	2	0.1	171763.0	171206.0	34	42.4	1.70%	0.00%
	2	0.2	171019.0	169222.0	126	733.9	2.60%	0.00%
	2	0.3	170957.0	169160.0	47	327.2	2.81%	0.00%
	4	0	270965.0	269042.0	1	0.1	0.00%	0.00%
	4	0.1	161098.0	157317.0	325	3965.1	2.40%	0.00%
	4	0.2	154898.0	154898.0	18	128.8	1.75%	0.00%
	4	0.3	158558.0	154898.0	12	210.4	1.89%	0.00%
pr1002	2	0	1148944.0	1104118.0	1	0.7	0.00%	0.00%
	2	0.1	914337.0	905223.0	3	13.4	0.15%	0.00%
	2	0.2	886438.0	886314.0	19	35.9	1.37%	0.00%
	2	0.3	886438.0	886314.0	18	24.6	1.21%	0.00%
	4	0	960821.0	960821.0	1	0.0	0.00%	0.00%
	4	0.1	682317.0	682317.0	1	44.8	0.00%	0.00%
	4	0.2	675993.0	675993.0	1	90.9	0.00%	0.00%
	4	0.3	675993.0	675993.0	1	63.6	0.00%	0.00%
Average					97.7	700.1	1.92%	0.76%

Table 7: Computational results for NCSP1' with $|V| = 30$.