Transforming Constraint Programs to Input for Local Search

On the relation between symmetry and neighborhoods

Jo Devriendt, Patrick De Causmaecker, and Marc Denecker

University of Leuven
{jo.devriendt,patrick.decausmaecker,marc.denecker}@cs.kuleuven.be

Abstract. Applying local search algorithms to combinatorial optimization problems is not an easy feat. Typically, human intervention is required to compile the constraints to input data for some metaheuristic algorithm. In this paper, we establish a link between symmetry properties of constraint optimization problems and local search neighborhoods, and we use this link to automatically generate neighborhoods from a constraint specification in the context of the IDP system. We evaluate the obtained neighborhoods for six classical optimization problems. The resulting observations support the viability of this technique.

1 Introduction

Combinatorial optimization problems are studied across many branches of science, and are abundant in industrial applications and real-life scenarios. One way to solve combinatorial optimization problems is to generate an initial, suboptimal solution, and to iteratively refine the solution until a stop-criterion is met and a hopefully optimal solution is achieved. This is roughly the process studied by the Metaheuristics community, which investigates techniques such as genetic algorithms, local search, hyperheuristics, swarm-based optimization etc. These techniques have proven very effective for many problems, especially for large problems with many (suboptimal) solutions.

The field of Constraint Programming (CP) also aims to solve combinatorial optimization problems effectively, but most CP systems perform a complete traversal of the search space by use of search trees with propagation, satisfiability solving, and/or mixed integer programming. These complete approaches are often effective, but potentially limit the size of the optimization problems that can efficiently be tackled.

In this paper, we investigate how we can bring both fields of research closer together by providing an automated way of transforming a constraint optimization problem specification into input for local search algorithms. The method is based on symmetry properties of optimization problems that allow to transform suboptimal solutions into hopefully better solutions.

This paper is organized in the following way. In Section 2, we give an abstract description of what a constraint optimization problem is, how it relates to certain forms of local search, and how symmetry can be used to derive input

neighborhoods for local search algorithms from a constraint optimization specification. The next section describes in more detail how this is done for a constraint optimization specification for the IDP. Section 4 reports on the nature of local search neighborhoods derived by the IDP system on multiple constraint optimization specifications. After subsequently sketching related work, the paper concludes.

2 Constraint Programming and Local Search

2.1 Constraint Satisfaction Optimization Problems

A COP can be characterized as a quadruple (V, D, C, O) where V denotes a set of variables, D a domain of possible values for these variables, C a set of constraints and O an objective function. An *assignment* to a CSP $\Pi = (V, D, C, O)$ is a function $\alpha : V \to D$. We refer to the set of all assignments to a COP Π as its *assignment space* As_Π.

We abstract a constraint $c \in C$ as the subset of As_Π for which c is satisfied. So, $\alpha \in c$ means α satisfies c, and $\alpha \notin c$ means α violates c. A *satisfying assignment* to a COP is an assignment which satisfies all constraints in C. We refer to the set of all satisfying assignments of a COP Π as the *solution space* Sol_Π. Note that the solution space is the intersection of all constraints – $Sol_\Pi = \cap_i c_i$ with $c_i \in C$ – and that a COP is unsatisfiable if the solution space is empty.

An objective function $O : (V \to D) \to \mathbb{N}$ maps assignments to natural numbers. Without loss of generality, we take an *optimal solution* to a COP to be a satisfying assignment that is minimal with respect to the objective function. More formally, a satisfying assignment $\alpha \in Sol_\Pi$ is an optimal solution if $\forall \alpha' \in Sol_\Pi : O(\alpha) \leq O(\alpha')$.

Example 1. A classical COP is the Traveling Salesman Problem (TSP). We can model this problem as a set of cities that need to be visited in a certain order, in such a way that the total distance of the visited tour is minimal. Given n cities, we can use $V = \{v_0, \ldots, v_{n-1}\}$ as a set of variables, the set of cities as domain D, and C containing the singular constraint that all variables must be assigned a different city. Given a distance matrix between cities $Dist : D \times D \to \mathbb{N}$, the objective function $O : \alpha \mapsto \sum_i Dist(\alpha(v_i), \alpha(v_{(i+1) \mod n}))$ maps each assignment to the sum of the distances between two subsequent cities.

2.2 Local search algorithm

Local search algorithms use the concept of a *neighborhood* to perform a heuristic walk through the solution space of a COP.

Definition 1. *A* neighborhood *N for a COP Π is a mapping of each satisfying assignment to a set of satisfying assignments $N : Sol_\Pi \to \mathcal{P}(Sol_\Pi)$. $N(\alpha)$ is referred to as the set of* neighbors *of a satisfying assignment α under N.*

Local search approaches such as those based on simulated annealing or tabu search require as input a neighborhood N and some initial satisfying assignment α. Given these, a typical local search algorithm explores the solution space by enumerating the neighbors of α under N. When some neighbor $\alpha' \in N(\alpha)$ satisfies an acceptance criterion (typically based on the objective value of α'), it is accepted and becomes the new focus of attention. In a sense, the local search algorithm *moves* from α to α', which is also expressed as executing a *move*.

Search continues by exploring the neighbors of α', until a new neighbor is accepted, leading to a new move, repeating the loop. This loop ends when some stop criterion is met, and the satisfying assignment with the lowest objective value encountered during the search is returned.

The above notion of local search is in a way restrictive, since it does not capture optimization approaches such as evolutionary programming or swarm-based optimization. Nonetheless, metaheuristic methods such as tabu search, simulated annealing, variable neighborhood search or greedy optimization can be characterized by moving from satisfying assignment to satisfying assignment using neighborhoods.

Example 2. For the TSP problem, a typical neighborhood is the so-called *2-opt* neighborhood. This neighborhood maps each TSP-tour α to a set of new tours by removing a pair of edges, say between cities c_1, c_2 and c_3, c_4, and reconnecting the resulting two TSP-subpaths by introducing an edge between c_1, c_4 and an edge between c_2, c_3.

2.3 Symmetry

Given the above definition of a neighborhood, it is interesting to investigate its relationship with the notion of symmetry. We follow [1] and define a symmetry of a COP as a permutation on the assignment space which preserves satisfaction to the constraints:

Definition 2. *A* symmetry S *for a COP* $\Pi = (V, D, C, O)$ *is a permutation on the assignment space* $S : (V \to D) \to (V \to D)$ *such that* $\alpha \in Sol_\Pi \Leftrightarrow S(\alpha) \in Sol_\Pi$.

By Definition 2, every permutation on the assignment space of an unsatisfiable COP is a symmetry. This makes Definition 2 a very general notion of symmetry, and in practice, only particular types of symmetry are considered. Some examples are *value symmetry*, *variable symmetry*, *row symmetry* and *column symmetry* [2]. More often than not, symmetries are induced by permutations on the set of variables or the set of values of a COP:

Definition 3. *A* variable symmetry *for a COP* (V, D, C, O) *is a symmetry* $S_\pi : \alpha \mapsto \alpha \circ \pi$ *induced by a permutation* $\pi : V \to V$. *A* value symmetry *is a symmetry* $S_\rho : \alpha \mapsto \rho \circ \alpha$ *induced by a permutation* $\rho : D \to D$.

Note that Definition 2 does not require a symmetry to be invariant with respect to the objective function; a symmetry S is *invariant* for objective function O iff $\forall \alpha : O(\alpha) = O(S(\alpha))$. When symmetry is used to reduce search time

by eliminating symmetric parts of the search space through *symmetry breaking*, the broken symmetry S must preserve the implicit minimization constraint of a COP. However, in a local search context, we will also investigate symmetries who are *variant* (as in *not invariant*) with respect to the objective function.

Example 3. The *Chromatic Number Problem* (CNP) consists of identifying the minimum number of colors with which a graph can be colored such that each two adjacent nodes have a different color. We can model this problem as a COP (V, D, C, O) where each node is a variable in V, each possible color a value in D, the constraints C state that two adjacent nodes in an input graph can not have the same color, and the objective function counts the number of colors used.

Any permutation ρ of the domain D induces a value symmetry S_ρ for the CNP. Each such S_ρ is invariant for the objective function.

Example 4. Using the TSP model from Example 1, each permutation of the domain induces a value symmetry and each permutation of the variables induces a variable symmetry. Both of these symmetry classes are variant for the objective function.

2.4 Symmetries induce a neighborhood

Note that the 2-opt neighborhood of Example 2 is based on a particular set of permutations of the set of cities in the TSP-tour. It is striking that these permutations also induce symmetries for the TSP-problem. We formalize this connection between symmetry and neighborhood:

Definition 4. *Given a set of symmetries $\mathcal{S}$ for some COP, the* symmetry-induced neighborhood $N_\mathcal{S}$ *maps each satisfying assignment α to its image under $\mathcal{S}$. More formally, $N_\mathcal{S} : \alpha \mapsto \{S(\alpha) \mid S \in \mathcal{S}\}$.*

By Definition 2, any satisfying assignment has only satisfying assignments as symmetry-induced neighbors, which ensures Definition 4 is a sound neighborhood definition.

Note however that Definition 4 requires some set of symmetries as input. Since a set of symmetries $\mathcal{S}$ forms a group $\Gamma_\mathcal{S}$ under functional composition, the number of possible symmetry sets to form neighborhoods with often is astronomical. In general, we have no definitive answer on what sets of symmetries one should use, but it seems plausible to use some small set $\mathcal{S}$ that generates the detected symmetry group $\Gamma_\mathcal{S}$. This way, each move possible under the induced neighborhood $N_{\Gamma_\mathcal{S}}$ can be simulated by a series of moves under $N_\mathcal{S}$, while $N_\mathcal{S}$ maps a satisfying assignment to a relatively small set of neighbors.

Using this notion of a symmetry-induced neighborhood, we can automatically compile a COP specification to input for a local search algorithm solving the COP. Recall that the only input required for many local search algorithms is some initial satisfying assignment α and a neighborhood N. The following steps generates these from only the problem specification:

1. Generate an initial satisfying assignment α using existing constraint programming technoloby.

2. Detect a symmetry group Γ of the constraint optimization problem.
3. Use some set of symmetries $\mathcal{S} \subseteq \Gamma$ to construct a symmetry-induced neighborhood $N_{\mathcal{S}}$.

3 Automating Local Search in IDP

We implemented the automatic detection of symmetry-induced neighborhoods in the COP-solving IDP system. In this section, we will sketch relevant details of the IDP system, as well as the type of symmetry and neighborhoods it detects. We also illustrate the detection by means of the TSP and CNP examples.

3.1 IDP as a constraint solving system

The IDP system is an experiment in constructing a *knowledge base* system. The aim of a knowledge base system is to solve problems in a radically declarative way, where domain knowledge is specified once as a knowledge base, allowing the user to solve multiple domain problems without further modifications to the knowledge base. The specification language of IDP is FO($\cdot$)– an extension of typed classical first-order logic with aggregates, arithmetic and inductive definitions.

One of the problems the IDP system is capable of solving is a logical *model optimization* problem (MOP). A MOP is the logical equivalent of a COP, which we will show after the brief introduction of some logical concepts.

In FO($\cdot$), logical formulas are constructed using a *vocabulary* Σ, which contains *type* symbols, *predicate* symbols and *function* symbols. The type symbols specify sets of *domain elements* present in the problem, while the predicate and function symbols specify respectively typed relations and typed functions. A *structure* I over a vocabulary Σ is an association of actual sets, relations and functions to the symbols in Σ. More formally, we say a structure I *interpretes* the symbols in Σ, while for a symbol $P \in \Sigma$, P^I is called its *interpretation*. Structures can be *partial*, in which case some predicate or function symbols have no interpretation. Given the symbols in a vocabulary, *formula*'s and *terms* can be constructed using logical connectives, quantifiers and other logical symbols. If a structure I ranges over the same vocabulary as a formula ϕ or a term t, then ϕ^I and t^I are evaluations of ϕ and t: ϕ^I is either true or false, while t^I maps to some domain element d from I.[1] A *theory* $\mathcal{T}$ over a vocabulary Σ is a set of formulas over Σ, and a structure I over Σ *satisfies* $\mathcal{T}$ if for all $\phi \in \mathcal{T}$, ϕ^I is true. In this case, we say that I is a *model* of $\mathcal{T}$, or $I \models \mathcal{T}$.

A MOP can be characterized as a quadruple $(\Sigma, J, \mathcal{T}, t)$, where Σ is the vocabulary, J a partial structure over Σ, $\mathcal{T}$ a theory over Σ and t a term over Σ mapping to a numeric domain such as the natural numbers. A MOP $(\Sigma, I, \mathcal{T}, t)$ then represents the task of finding

[1] To be exact, ϕ and t should also not contain any unquantified logical variables to have a proper evaluation in I.

– a model $I \models \mathcal{T}$,
– such that I has the same interpretation as J for all types and interpreted symbols in J,
– and I is minimal for t.[2]

When relating this to a COP (V, D, C, O), $\mathcal{T}$ represents the constraints C, t represents the objective function O, the uninterpreted function and predicate symbols in J represent the variables V, and the domain of values D corresponds to the set of possible relations and functions that can be associated to the uninterpreted symbols, given the interpretation of the types of Σ in J. A structure corresponds to an assignment, a model to a satisfying assignment, and a minimal model to an optimal solution.

Instead of diving into the technical details of FO$(\cdot)$, we improve a reader's intuition by the following two examples:

Example 5. The COP specification for TSP given in Example 1 can be realized as a MOP with the following vocabulary:

– type $City$
– type $Index \subseteq \mathbb{N}$
– function symbol $Distance : City \times City \to \mathbb{N}$
– function symbol $Next : Index \to Index$
– function symbol $Map : Index \to City$

For the TSP, every symbol but Map will be interpreted by the partial structure, so Map will function as our search variable.

The TSP constraints are specified by a theory with one formula:

$$\forall x : \forall y : (Index(x) \wedge Index(y) \wedge x \neq y) \Rightarrow Map(x) \neq Map(y).$$

Which effectively posts an all-different constraint over the Map function symbol, stating that two different index elements need to be mapped to two different cities. Each model satisfying this theory will thus have a TSP tour as interpretation for Map.

Next we need a minimization term:

$$\sum_{z} \{Distance(Map(z), Map(Next(z))) \mid z \in Index\}$$

Which denotes the sum of all distances between cities mapped by subsequent indices, and as such denotes the total distance of a tour represented by the Map function symbol.

Finally, the partial structure provides the necessary parameters to solve a TSP instance. For this, it contains

– a set of indices $\{0, \ldots, n-1\}$ and a set of cities $\{c_1, \ldots, c_n\}$ as interpretation for the types,
– a function adhering to the signature of the $Distance$ symbol specified in the vocabulary,

[2] Again we assume the objective function is to be minimized.

– a function adhering to the signature of the $Next$ symbol specified in the vocabulary, which for sensible TSP instances maps some index x to $(x + 1)$ mod n.

Given the above vocabulary, theory, minimization term and partial structure, IDP's MOP routine then searches for an interpretation to Map that satisfies the constraints in the theory, and minimizes the objective function. This solves the TSP COP described in Example 1, as the inferred interpretation of Map leads to an optimal TSP tour.

Example 6. The CNP from Example 3 can be modelled in FO(·) using the following vocabulary:

– type $Node$
– type $Color$
– predicate symbol $Edge \subseteq Node \times Node$
– function symbol $Coloring : Node \to Color$

The constraint that two neighboring nodes must have a different color is stated in the following theory:

$$\forall x : \forall y : Edge(x, y) \Rightarrow Coloring(x) \neq Coloring(y).$$

And the objective function simply counts the number of colors used:

$$\#\{z \mid \exists x : Coloring(x) = z\}$$

Finally, a partial structure contains an input graph by interpreting $Edge$, and leaves the $Color$ symbol uninterpreted. Solving the MOP will lead to an interpretation for $Color$, representing a minimal graph coloring.

Algorithmically, IDP solves model optimization by a *ground-and-solve* approach, where the theory, structure and minimization term are *grounded* to a set of low-level constraints in the *extended conjunctive normal form* (ECNF) language. These ECNF constraints can be solved by IDP's custom-made *lazy clause generation* constraint solver MiniSAT(ID) [3]. This grounding step is analogous to the *flattening* of high-level MiniZinc specifications to FlatZinc constraints, to the point that MiniSAT(ID) can also solve FlatZinc specifications.[3]

3.2 Symmetry in FO(·)

Given that we defined a symmetry as a permutation of the assignment space preserving satisfaction to the constraints, a symmetry of a MOP $(\Sigma, J, \mathcal{T}, t)$ corresponds to a permutation of the set of extensions of J such that for each extension I of J holds $I \models \mathcal{T} \Leftrightarrow S(I) \models \mathcal{T}$.

The symmetry type detected by IDP is *domain element swap* (DES) symmetry, which is a variant of the type of symmetry detected by the relational

[3] MiniSAT(ID) also participated in 2014's and 2015's FlatZinc competition.

model finder Kodkod [4]. Kodkod uses untyped first-order logic, and hence provides only one set of domain elements Dom in its partial structure J. Kodkod's symmetry detection routine then partitions Dom into subsets Dom_i such that for any Dom_i all swaps of domain elements $d_1, d_2 \in Dom_i$ induce a symmetry. IDP's DES symmetry similarly exploits the given types as partition of the set of all domain elements, but it does not require all symbols ranging over the type to take part in the symmetry.

Let's provide a formal definition for clarification:

Definition 5. *A* domain element swap *(DES) symmetry S of a MOP $(\Sigma, J, \mathcal{T}, t)$ is a symmetry characterized by a triple (a, b, σ) such that a, b are two domain elements from the same type interpretation in J, and σ is a subset of predicate and function symbols from Σ that are uninterpreted in J. If we take π_{ab} to be the permutation of domain elements that swaps a with b and leaves all other domain elements invariant, then S maps each extension I of J to $S(I)$ in such a way that for predicate symbols $P \in \sigma$:*

$$(d_1, \ldots, d_n) \in P^I \Leftrightarrow (\pi_{ab}(d_1), \ldots, \pi_{ab}(d_n)) \in P^{S(I)}$$

and for function symbols $f \in \sigma$:

$$d_0 = f^I(d_1, \ldots, d_n) \Leftrightarrow \pi_{ab}(d_0) = f^I(\pi_{ab}(d_1), \ldots, \pi_{ab}(d_n))$$

while the interpretation of predicate symbols $Q \notin \sigma$ and function symbols $g \notin \sigma$ is left untouched:

$$Q^I = Q^{S(I)} \text{ and } g^I = g^{S(I)}$$

The following two examples illustrate DES symmetry:

Example 7. The TSP MOP from Example 5 exhibits two classes of DES symmetry:

- DES symmetries characterized by $(i, j, \{Map\})$ for $i, j \in [0 \ldots n-1]$. These symmetries swap two indices i and j in the interpretation of Map, and as such are equivalent to the TSP variable symmetry of Example 4.
- DES symmetries characterized by $(c_i, c_j, \{Map\})$ for $i, j \in [1 \ldots n]$. These symmetries swap two cities c_i and c_j in the interpretation of Map, and as such are equivalent to the TSP value symmetry of Example 4.

Example 8. The chromatic number MOP from Example 6 exhibits the following class of DES symmetry:

- DES symmetries characterized by $(c, c', \{Coloring\})$ for two domain elements c, c' in $Color$'s interpretation. These symmetries swap two colors c and c' in the interpretation of $Coloring$, and as such are equivalent to the chromatic number value symmetry of Example 6.

3.3 Symmetry-induced neighborhood detection in IDP

As mentioned at the end of Section 2, we can convert a COP specification to input for a local search algorithm by use of a symmetry-induced neighborhood. The previous section explained the type of symmetry the IDP system can detect for a MOP, so all that is left to do is figure out which of these symmetries induce good neighborhoods.

We put forward that symmetries that are invariant for the objective function are bad candidates for neighborhood generation. Often such symmetries constitute a simple renaming of variables, values or domain elements, and as such can not transform a satisfying assignment into a reasonably different one. Of course, sometimes a move in a local search algorithm transforms the current satisfying assignment into one with the same objective value, but having a neighborhood that *only* leads to such moves seems like a waste of resources. We shortly investigate some properties of DES symmetry present in a MOP problem with regard to the invariance of the objective function.

Firstly, a sufficient condition for a DES symmetry (a, b, σ) to leave the minimization term t invariant is that σ does not contain any predicate or function symbols occurring in t. As such, we can instruct IDP's neighborhood detection algorithm to only investigate symmetries ranging over some uninterpreted symbol in t. For the TSP problem specification from Example 5 the optimization term contains the uninterpreted symbol Map, which occurs in both symbol lists of the TSP DES symmetries given in Example 7. As a result, it is possible that both symmetry classes are not invariant for the minimization term, which upon further inspection is the case.

However, ranging over an uninterpreted symbol in the minimization term is not a necessary condition for a DES symmetry to be invariant for the minimization term, as shown by the CNP. The CNP MOP minimization term (see Example 6) contains the function symbol $Coloring$, which also occurs in the list of symbols of the DES symmetry given in Example 8. However, swapping two colors in a graph coloring is invariant for the number of colors used. IDP's symmetry detection scheme utilizes more refined mechanisms to detect whether terms and formula's are invariant under a certain DES symmetry, which we will use in the experiments, but which will not be explained in further detail here.

With the above points in mind, we can devise a simple symmetry-induced neighborhood detection scheme for a MOP $(\Sigma, J, \mathcal{T}, t)$ in IDP:

1. Identify the predicate and function symbols occurring in t but uninterpreted in J.
2. Detect DES symmetry over these symbols.
3. Ignore any DES symmetry invariant for t.
4. Convert the remaining DES symmetries to neighborhoods by Definition 4.

The performance critical part of this algorithm is the symmetry detection step, whose efficiency in turn depends on the granularity of the symmetry detected. For IDP, symmetry detection takes at most $O(n^2)$ time, with n the total

number of domain elements in J, since worst-case it generates all pairs of domain elements a, b to check for DES symmetries (a, b, σ). The list of symbols σ is derivable in linear time from $\mathcal{T}$. So for DES symmetries, the neighborhood detection mechanism is tractable.

The only question that remains is what (small) set of symmetries should be used to induce neighborhoods, as was mentioned at the end of Section 2.4. Note that DES symmetries represent swaps of domain elements, which can be composed to form other symmetries based on other permutations of domain elements. Moreover, any permutation over a set of domain elements can be obtained by a composition of swaps of domain elements. As a result, detecting a set $\mathcal{S}$ of DES symmetries entails detecting a group of symmetries $\Gamma_{\mathcal{S}}$ that represents the interchangeability of subsets of domain elements (those that can be pairwise swapped).

The size of $\Gamma_{\mathcal{S}}$ is factorial in the size of $\mathcal{S}$, so using all symmetries in $\Gamma_{\mathcal{S}}$ as neighborhood inducing symmetries seems an infeasible option. Instead, we restrict the set of neighborhood inducing symmetries to the set of all possible swaps of domain elements, in casu $\mathcal{S}$. This has two advantages: firstly it limits the amount of neighborhood inducing symmetries to $O(n^2)$ with n the number of swappable domain elements. Secondly, $\mathcal{S}$ *generates* $\Gamma_{\mathcal{S}}$, meaning that any model $S(I)$ that can be reached by $S \in \Gamma_{\mathcal{S}}$ from model I can also be reached by a composition of some series of $S' \in \mathcal{S}$.

On the other hand, it is possible to construct a set of symmetries $\mathcal{S}'$ that generates $\Gamma_{\mathcal{S}}$ but which is $O(n)$ in size. $\mathcal{S}'$ then consists of swaps of subsequent domain elements d_i, d_{i+1} according to some chosen total order on the domain elements. Even though it would lead to smaller neighborhoods, it also skews any local search algorithm according to the chosen order, putting a possibly unwarranted bias on the direction of the search over the solution space. For this reason, we stick with the quadratic set of symmetries $\mathcal{S}$ to induce a neighborhood.

4 Experiments

We implemented the neighborhood detection scheme described in the previous section in IDP, and in this section we experimentally investigate which neighborhoods were detected for a series of well-known constraint optimization problems. The FO$(\cdot)$ specifications for each of these problems are available online at `adams.cs.kuleuven.be/idp/localsearch.html`[4], where an interested reader can run IDP with one click and see the neighborhood detection mechanism in action.

4.1 Traveling Salesman Problem: on robustness

Let us first investigate the TSP problem, since this was our running example. As mentioned in Section 3, both the city-swapping and index-swapping symmetries

[4] Click "File", then "Local Search in IDP"

are variant for the minimization term, resulting in a neighborhood swapping cities and indices.

However, there exist other reasonable FO($\cdot$) specifications of TSP other than the one in Example 5. For instance, a user could use the following vocabulary, theory and minimization term specifying the TSP:

Vocabulary:

- type $City$
- function symbol $Distance : City \times City \rightarrow \mathbb{N}$
- predicate symbol $Following \subseteq City \times City$
- predicate symbol $Reachable \subseteq City$
- constant $Start : \rightarrow City$

Theory:

$$\forall x : \exists 1 y : Following(x, y).$$
$$\forall y : \exists 1 x : Following(x, y).$$
$$\forall x : Reachable(x).$$
$$\{\forall x : Reachable(x) \leftarrow x = Start \vee (\exists y : Reachable(y) \wedge Following(y, x)).\}$$

The constraint between curly brackets is an *inductive definition* [5], constraining the *Reachable* predicate to only be true for cities reachable from a *Start* city using the *Following* relation. By next stating that all cities must be reachable, we effectively posted a subtour elimination constraint.

Minimization term:

$$\sum_{x,y} \{Distance(x, y) | Following(x, y)\}$$

When executing the described neighborhood detection algorithm, we establish that *Following* is an uninterpreted predicate symbol occurring in the objective function, and that DES symmetries swapping cities over this symbol exist. Also, these symmetries are variant for the objective function, so they lead to a city-swapping neighborhood.

This alternative specification experiment shows that the proposed neighborhood detection method exhibits robustness: different specifications of the same problem still lead to comparable neighborhoods. This of course only holds as long as symmetry properties of different specifications are similar.

4.2 Shortest Path Problem: a succesful problem

This problem consists of finding the shortest path between a start and end node in a weighted graph. Its specification in FO($\cdot$) is very similar to the TSP specification of the previous subsection, and centers on finding a minimal interpretation to some $Following/2$ predicate. IDP's neighborhood detection mechanism

indeed detected that all two cities except the start and end city were interchangeable. The resulting symmetries were variant for the objective function, leading to a large set of induced neighborhoods. Therefor, we judge IDP's neighborhood detection algorithm to be succesful on the shortest path problem.

4.3 Max Clique: relaxing constraints?

The task of a max clique problem is to identify the largest clique in a graph. We modelled this problem in such a way that each satisfying assignment represented a set of nodes forming a clique in the input graph. The only domain elements in this problem are the nodes of the graph, and for typical graphs nodes are not swappable. Hence, IDP could not detect any symmetry, and no induced local search neighborhoods were detected by our algorithm. This makes sense, since there is no obvious way of transforming cliques of a graph in one move to some other clique in the graph.

However, we can imagine a local search algorithm for this problem that iteratively removes nodes from and to a node set representing a potential clique. This effectively *relaxes* the clique constraint, making any set of nodes a satisfying assignment, regardless of whether it forms a clique in the input graph. The lesson we take from this problem is that our neighborhood detection scheme is not yet able to detect constraints that can be relaxed to allow for a larger potential neighborhood.

4.4 Chromatic Number Problem: avoiding a useless neighborhood

As mentioned in Example 3, the CNP consists of finding the lowest amount of colors with which a graph can be colored so that no two adjacent nodes have the same color. This problem does exhibit symmetry in the sense that all colors to color the nodes with are swappable. However, as mentioned in Section 3.3, this symmetry is invariant for the objective function, and as a result, no symmetry-induced neighborhood was detected. This is a positive result, since globally swapping colors does not lead to an effective local search neighborhood.

However, swapping colors for each node separately might lead to a reasonable neighborhood, which would again relax the constraint that two adjacent nodes must have a different color. This observation is similar to the one made in the Max Clique section.

4.5 Knapsack Problem: unexpected symmetry leads to unexpected neighborhood

The knapsack problem consists of filling up some abstract knapsack with objects, such that the volume of the objects fits the knapsack, but the value of the objects is maximal. Since swapping any two objects in and out of the knapsack might violate the volume constraint, we expected the neighborhood detection algorithm to not detect any symmetry, and thus no neighborhood.

However, IDP's symmetry detection was sufficiently fine-grained to identify that for some instances, some objects had the same volume but a different value. These objects could safely be swapped in and out of the knapsack, leading to an unexpected neighborhood where for a given knapsack, small variations on the filling of the knapsack could be explored.

4.6 Assignment Problem: human neighborhoods

The last problem we investigate is the assignment problem, where a bijection between a set of agents and a set of tasks must be found that minimizes the cost of assigning a certain task to a certain agent. As expected, IDP detected that swaps of agents and tasks were not a priori invariant for the objective function. As a result, these symmetries induced a local search neighborhood, which arguably would be the same neighborhood a human algorithmician would devise. This is an optimal neighborhood detection result!

4.7 Experimental conclusions

Testing the symmetry-based automated neighborhood detection algorithm of IDP on the above problems yielded interesting insights. Firstly, the approach is robust in changes to the specification as long as the symmetry properties are not disturbed. Secondly, taking invariantness of the minimization term into account allows to avoid detecting useless neighborhoods. Thirdly, for some problems, the symmetry-induced neighborhood is the same as the one a human would devise. Fourthly, to detect more neighborhoods, it might be needed to relax constraints. And finally, sometimes the automated neighborhood detection algorithm might find neighborhoods where a human did not expect them.

On the whole, we evaluate the experiment to have returned positive results, supporting the viability of symmetry-induced neighborhood detection.

5 Related Work

In the previous sections we described how any constraint programming system and how IDP in particular can exploit symmetry properties of problems to derive local search neighborhoods for those problems. Of course, we are not the first to try to link local search to constraint programming. A well-known example is the Comet system, which allows a user to easily specify neighborhood based local search algorithms in a constraint-centered way [6, 7]. However, Comet did not provide any automatic neighborhood detection algorithm.

Stochastic SAT solvers such as WalkSAT [8] also allow a constraint programming problem to be solved by local search. In WalkSAT, every assignment to the boolean variables is a satisfying assignment, and neighborhoods are defined in terms of "flips" on the truth value of a boolean variable. In our view, this is an extreme approach, since all original constraints can be violated by any move. Note that in a context where all constraints are relaxed, any permutation on the

solution space is a symmetry which can be used for symmetry-induced neighborhoods. However, these neighborhoods are not the kind a human programmer would devise for most combinatorial optimization problems.

To our knowledge, the only system that allows automatic derivation of local search neighborhoods from an input specification is LocalSolver [9]. LocalSolver allows a user to write down constraints in a mathematical modeling language centered on boolean variables, and uses flips on those variables as well as feasibility preserving moves based on "ejection chains applied to the hypergraph induced by boolean variables and constraints". We conjecture that these feasibility preserving moves can be seen as symmetries of the problem, but we need further investigation to confirm this. One weakness of LocalSolver is that the initial solution is found by a basic randomized greedy algorithm, and that it is not designed for solving hardly-constrainted optimization problems. This is opposed to the constraint-based local search approach described in this paper, which can always fall back on the solving capabilities of a state-of-the-art constraint programming engine.

To put our work in a broader perspective, it is worth mentioning that much research has been done on the relationship between symmetry *breaking* and local search (e.g. [10]). An important result is that adding symmetry breaking constraints has a negative impact on the efficiency of local search algorithms. In our work, we do not break any symmetry, but rather exploit symmetries to traverse the local search space. Most importantly, the symmetries we use are not symmetries of the whole optimization problem, since they are variant for the objective function. As such, they cannot be broken, since this would risk removing an optimal solution from the search space. In this light, symmetry breaking for optimization problems and exploiting symmetry properties to construct local search neighborhoods are two orthogonal uses of symmetry.

On the other hand, neighborhood inducing symmetries share the fact that they are variant for the objective function with *dominance relations* used in *dominance breaking*. Dominance breaking is a generalization of symmetry breaking for complete search algorithms where extra constraints remove *dominated* solutions with a worse objective value from the search space [11]. These dominance relations seem to also induce neighborhoods much in the same way symmetry does and can be detected automatically [12], but it is not clear if dominance relations of a COP are fundamentally different to symmetries that are variant for the objective function.

Finally, in recent years the metaheuristic community is actively investigating how to formalize and automate local search algorithms [13]. Our work can be seen as an effort in that direction, opening up unexplored research avenues by linking local search neighborhoods to symmetry properties of problems.

6 Conclusion and future work

In this paper, we propose a link between local search neighborhoods and symmetry. To our knowledge, this is the first time such a link is established. We used

this link to design and implement a transformation of a COP specification, in the form of a MOP specified in FO($\cdot$), to input for an automated local search algorithm. We conducted an experimental investigation of the symmetry-induced neighborhoods detected by our implementation, which supports the viability of this technique.

As future work, it remains to be seen how well symmetry-based neighborhood detection algorithms perform on larger, more complex problems, with potentially more symmetry breaking constraints. Allowing the relaxation of certain constraints might prove crucial in detecting sufficiently large neighborhoods. Also, the link with techniques for dominance breaking is definitely worth investigating.

Secondly, it would be interesting to couple the detected neighborhoods and input solutions to some local search engine, and experimentally verify their performance. The challenge here lies in being able to quickly move from one satisfying assignment to another, and to incrementally update the value of an objective function. The IDP system is currently unoptimized in this regard.

Thirdly, the problem of deciding which subset of detected symmetries are used to induce neighborhoods can also be tackled by a *hyperheuristic* algorithm. In general, a hyperheuristic aims to find the best combination of neighborhoods by deciding which neighborhood to select for the next iterations of local search and evaluating the resulting executed moves. Providing information on the whole symmetry group to the hyperheuristic algorithm instead of only a subset of symmetry-induced neighborhoods might allow it to derive more fitting combinations of neighborhoods.

Lastly, for some problems, there exist complex neighborhoods involving smart perturbation and repair steps. Detecting these still seems a hard challenge.

Acknowledgements

Thanks go to Ingmar Dasseville for configuring the IDP web interface with our local search examples, and to Bart Bogaerts and the anonymous reviewers for helpful comments that improved this paper.

References

1. Cohen, D., Jeavons, P., Jefferson, C., Petrie, K.E.: Symmetry definitions for constraint satisfaction problems. In: Constraints. (2006) 115–137
2. Van Hentenryck, P., Flener, P., Pearson, J., Ägren, M.: Compositional derivation of symmetries for constraint satisfaction. In Zucker, J.D., Saitta, L., eds.: Abstraction, Reformulation and Approximation. Volume 3607 of LNCS. Springer Berlin Heidelberg (2005) 234–247
3. De Cat, B., Bogaerts, B., Devriendt, J., Denecker, M.: Model expansion in the presence of function symbols using constraint programming. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013, IEEE Computer Society (2013) 1068–1075
4. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In Grumberg, O., Huth, M., eds.: TACAS. Volume 4424 of LNCS., Springer (2007) 632–647

5. Denecker, M., Ternovska, E.: A logic of nonmonotone inductive definitions. ACM Trans. Comput. Log. **9**(2) (April 2008) 14:1–14:52
6. Michel, L., Van Hentenryck, P.: The Comet programming language and system. In van Beek, P., ed.: CP. Volume 3709 of LNCS., Springer (2005) 881–881
7. Van Hentenryck, P., Michel, L.: Constraint-Based Local Search. MIT Press (2005)
8. Selman, B., Kautz, H., Cohen, B.: Local search strategies for satisfiability testing. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science. (1995) 521–532
9. Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: Localsolver 1.x: a black-box local-search solver for 0-1 programming. 4OR **9**(3) (2011) 299–316
10. Prestwich, S., Roli, A.: Symmetry breaking and local search spaces. In Bartk, R., Milano, M., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 3524 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 273–287
11. Chu, G., Stuckey, P.J.: A generic method for identifying and exploiting dominance relations. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming. CP'12, Berlin, Heidelberg, Springer-Verlag (2012) 6–22
12. Mears, C., García de la Banda, M.: Towards automatic dominance breaking for constraint optimization problems. In Yang, Q., Wooldridge, M., eds.: IJCAI, AAAI Press (2015) 360–366
13. Swan, J., Adriaensen, S., Bishr, M., Burke, E.K., Clark, J.A., De Causmaecker, P., Durillo, J., Hammond, K., Hart, E., Johnson, C.G., Kocsis, Z.A., Kovitz, B., Krawiec, K., Martin, S., Merelo, J.J., Minku, L.L., Özcan, E., Pappa, G.L., Pesch, E., García-Sánchez, P., Schaerf, A., Sim, K., Smith, J.E., Stützle, T., Vo, S., Wagner, S., Yao, X.: A research agenda for metaheuristic standardization. In: Proceedings of the XI Metaheuristics International Conference, June 7-10, 2015, Agadir, Morocco. (2015)