EPiC Series in Computing
Volume 41, 2016, Pages 266–279
GCAI 2016. 2nd Global Conference on Artificial Intelligence

# Matching Qualitative Constraint Networks with Online Reinforcement Learning

Malumbo Chaka Chipofya[1]

Institut für Geoinformatik
Westfälische Wilhelms-Universität Münster
Münster, Germany
`mchipofya@uni-muenster.de`

**Abstract**

Local Compatibility Matrices (LCMs) are mechanisms for computing heuristics for graph matching that are particularly suited for matching qualitative constraint networks and thus enabling the transfer of qualitative spatial knowledge between qualitative reasoning systems or agents. A system of LCMs can be used during matching to compute a pre-move evaluation, which acts as a prior optimistic estimate of the value of matching a pair of nodes, and a post-move evaluation which adjusts the prior estimate in the direction of the true value upon completing the move. We present a metaheuristic method that uses reinforcement learning to improve the prior estimates based on the posterior evaluation. The learned values implicitly identify unprofitable regions of the search space. We also present data structures that allow a more compact implementation, limiting the space and time complexity of our algorithm.

## 1 Introduction

Qualitative spatial relations such as those defined in the $\mathcal{RCC}8$ calculus [6] have proven useful for producing descriptions of spatial scenes that can be processed by computer programs in contexts where precise numerical representations are either too expensive or too expressive for the task at hand. An important task for some applications that use qualitative information is the projection of a qualitative description from some source such as a textual description or a sketch map onto a target such as a spatial database (see e.g. [8, 9]). Particular examples of such applications include the SketchMapia framework for sketch based GIS interaction [20], human-robot interaction systems [21], and knowledge transfer between artificial agents for cooperation or learning [10].

When the qualitative descriptions are given as graphs the process of finding the best projection is called graph matching. In this work we are interested in a particular type of graph matching where the graphs being matched are qualitative constraint networks (QCNs) [19]. This is known as the QCN matching problem (QMP) [5].

Despite the existence of the applications cited above, the QCN matching problem (QMP) has not been addressed adequately. The design of a scalable approach for QMP remains an open problem. A matching algorithm for QMP that took into account the semantics of the spatial relations being considered was presented in [23]. Their method is based on a semi-formal definition of QMP and uses an approximation of path-consistency, called algebraic closure, to produce heuristics for exploring the search space. We gave a formal definition of QMP in [4] and [5] based on work in [23]. In [4] we used a metaheuristic approach to solve small instances of QMP.

The approach by [23] mentioned above does not scale to realistic use cases such as, for example, the case of geographic scale sketch maps with 20 to a few hundred spatial entities. Our metaheuristic approach in [3] on the other hand has been applied to larger sketch maps with up to 80 objects. However even the metaheuristic method we use there suffers from long-term forgetfulness. If a solution is revisited after many intervening solutions, it is likely that the same actions will be repeated.

In the present work we outline an approach for solving QMP that incorporates learning and could therefore be more scalable with respect to input data sizes and variations of the matching problem than previous methods. In particular we give a technical interpretation of the so called local compatibility matrix (LCM) that allows a compact, information rich representation of the state space of a QMP instance [3]. A QMP instance is associated with a system of LCMs that can be used to efficiently compute a pair of heuristic evaluation functions $\mathtt{e}_{(\cdot,\cdot)|m}$ and $\mathtt{e}_m$.

We use $\mathtt{e}_{(\cdot,\cdot)|m}$ and $\mathtt{e}_m$ to learn better heuristic estimates for exploring the search space. The learning takes the form of SARSA updates [22] from Reinforcement Learning (RL). To evade the curse of dimensionality, we have devised an effective prototype based state-action value representation for the SARSA learner.

Our presentation in this paper is motivated by practical considerations. We intend to exemplify how learning approaches can be employed in solving the QCN matching problem. To this end, our presentation is intended to be clear enough that other researchers can implement it. Our evaluation is performed on the sketch-to-metric map alignment problem which is of interest to a wider community and for which data is readily available.

### 1.1 Sketch-to-metric Map Alignment

Sketch-to-metric map alignment is the process of grounding sketch maps within a (plani)metric map by identifying which features in the sketch map correspond to which ones in the metric map. The goal of sketch-to-metric map alignment is to establish correspondences of objects identified in the input sketch map to objects in the metric map. Applications of sketch-to-metric map alignment span all the geospatial examples cited above.

### 1.2 QCN Matching

A QCN, also known as a *complete* qualitative constraint graph, is a graph $\mathcal{N} = (N, l)$, where $N$ is a set of nodes and $l : N \times N \to 2^{\mathbf{R}}$ is a labelling function which assigns to every pair $i, j \in N$ a label $R \subset \mathbf{R}$ called the constraint on $i, j$. Typically, the labels are drawn from the powerset of atomic relations over a spatial domain (e.g. the set of $\mathcal{RCC}8$ relations: $DC$, $EC$, $PO$, $EQ$, $TPP$, $TPPI$, $NTPP$, and $NTPPI$). A label $R'$ is said to be a refinement of a label $R$ if $R' \subset R$ and a QCN $Q$ is said to be refined to a QCN $Q'$, written $Q' \leq Q$, if any of $Q$'s labels are replaced by any of their refinements to obtain $Q'$.

A QCN $\mathcal{N}$ is said to be consistent if there exists an assignment of objects from the spatial domain being represented to the nodes of $\mathcal{N}$ such that for every pair of the objects the relation containing the pair is an element of the constraint on the corresponding pair of nodes. A QCN is algebraically closed (or simply, closed) if every restriction to three nodes is consistent [15, 14].

The disjoint union of $\mathcal{N}$ and $\mathcal{N}' = (N', l')$ is $\mathcal{N} \uplus \mathcal{N}' = (N \cup N', l'')$ where

$$l''(i,i') = \begin{cases} l(i,i') & \text{if} \quad i,i' \in N \\ l'(i,i') & \text{if} \quad i,i' \in N' \\ \mathbf{R} & \text{otherwise} \end{cases}$$

An arc from $N$ to $N'$ is called a joining arc. Let **Id** be the identity relation in **R**. That is, **Id** consists of object pairs $(a, b)$ that cannot be distinguished by the relations in **R**. Let $\mathbf{I}_{\mathcal{N}}$ denote the arcs of $\mathcal{N}$ labelled with **Id**. The QCN matching problem is the problem of finding refinements of the joining arcs that maximize the number of those arcs labelled with the identity relation such that the resulting QCN is algebraically closed. Where algebraic closure decides consistency, this formulation of QMP guarantees that a solution is still attainable even in the presence of uncertain information.

**Definition 1.** *$QMP(\mathcal{N},\mathcal{N}')$: find $\mathcal{M} \leq \mathcal{N} \uplus \mathcal{N}'$ such that*

1. *$\mathcal{M}$ is algebraically closed, and*
2. *for any closed $\mathcal{M}' \leq \mathcal{N} \uplus \mathcal{N}'$, $|\mathbf{I}_{\mathcal{M}'}| \leq |\mathbf{I}_{\mathcal{M}}|$.*

The relationships between QMP and the maximum common subgraph problem [12] as well as experience from [16, 2, 23, 4] suggest that QMP is a difficult problem and it is doubtful that polynomial algorithms can be devised for it. This has motivated work into heuristic and metaheuristic solutions to QMP including the present work.

In this work we consider a simple local search scheme with a prohibition mechanism known as a tabu list [11]. At each iteration our algorithm can perform one of two actions: an ADD move adds a pair of matched nodes to the current solution and a DROP move removes a pair from the current solution.

The remainder of this paper is organised as follows. We first introduce LCMs and the two heuristic functions in Section 2. The extension to learning is detailed in Section 3. In Section 4 we describe an efficient algorithm and data structures for representing and exploiting LCMs in QCN matching algorithms followed by a brief evaluation of our method in Section 5. We close the paper with some concluding remarks in Section 6.

## 2 LCM based Heuristics for QCN Matching

Local compatibility matrices [3] are an alternative to the standard 0–1 compatibility matrices [7] which make explicit use of the edge labels from the input graphs. The idea is to obtain a more general representation of compatibility between pairs of nodes that supports direct exploitation of labels in the input graphs during the matching itself.

### 2.1 Local Compatibility Matrices

A compatibility matrix for the matching problem $QMP(\mathcal{N}, \mathcal{N}')$ has for every two pairs $(i, i')$ and $(j, j')$ of nodes, with $i, j \in N$ and $i', j' \in N'$, two cells containing the value $c[l(i, j), l(i', j')]$, for some compatibility function[1] $c[\cdot,\cdot]$ on $2^{\mathbf{R}}$. Rows and columns are indexed by the node pairs such that $c[l(i, j), l(i', j')]$ is the value at the intersections of row $(i, i')$ and column $(j, j')$ and vice-versa. Thus there are $|N \times N'|^2$ entries in the compatibility matrix.

[1]Usually a similarity function based on some metric $d[\cdot,\cdot] : 2^{\mathbf{R}} \times 2^{\mathbf{R}} \to [0, 1]$ is used. E.g. $c[x,y] = 1 - d[x,y]$.

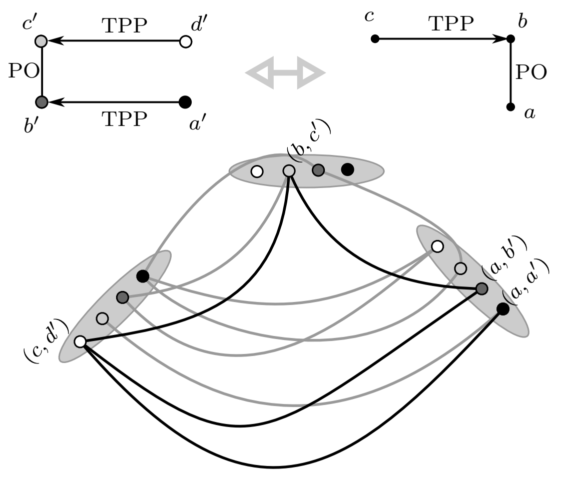


Figure 1: Two $\mathcal{RCC}8$ QCNs to be matched and the corresponding association graph created by drawing an edge between two nodes $(i,i')$ and $(j,j')$ whenever $l(i,j) \cap l(i',j') \neq \emptyset$. The label on non-edges is the relation $\mathcal{RCC}8$ relation $DC$. In the association graph a node is a pair of a circle and the oval containing it.

In a LCM there is a special *reference pair* of nodes with respect to which the matrix is defined. If $(i,i')$ is the reference pair the matrix will have entries $l(i,j) \cap l(i',j')$ for every $j \in N$ and $j' \in N'$. The LCM expresses compatibility between the reference pair $(i,i')$ and every other pair $(j,j') \in N \times N'$. Rows are indexed by input nodes $j \in N$ and columns are indexed by target nodes $j' \in N'$. The entry in a cell is the largest label common to both edges $l(i,j)$ and $l'(i',j')$. Each LCM has $|N \times N'|$ entries and there are $|N \times N'|$ LCMs for $QMP(\mathcal{N}, \mathcal{N}')$.

To illustrate how LCMs work we consider the simple example matching problem in Figure 1. The association graph in the lower part of the figure shows which node pairs are consistent with each other in terms of the label intersections. The following LCM with reference pair $(c,d')$ corresponds to the black arcs in the association graph:

|  | $a'$ | $b'$ | $c'$ | $d'$ |
|---|---|---|---|---|
| $(c,d')$ | $(DC)$ | $(DC)$ | $(TPP)$ | (equal) |
| $a(DC)$ | $DC$ | $DC$ | $\emptyset$ | $\emptyset$ |
| $b(TPP)$ | $\emptyset$ | $\emptyset$ | $TPP$ | $\emptyset$ |
| $c$(equal) | $\emptyset$ | $\emptyset$ | $\emptyset$ | equal |

## 2.2 Heuristic Evaluations

LCMs have the nice property that, for non-overlapping labels, cells having the same label form a submatrix of the LCM and that these submatrices are non-overlapping [3]. So, the rows and columns of an LCM can be sorted such that cells with a particular label are in a contiguous region of the LCM. This led to the discovery of the two heuristic evaluation functions based on LCMs.

Let $\mathcal{L}_{(i,i')}$ be the LCM with reference pair $(i,i')$ and $\mathcal{L}_{(i,i'),R}$ the submatrix of $\mathcal{L}_{(i,i')}$ with label $R \subseteq \mathbf{R}$. Let $\mathsf{rows}(\mathcal{L}_{(i,i'),R})$ and $\mathsf{cols}(\mathcal{L}_{(i,i'),R})$ denote the number of rows and columns of $\mathcal{L}_{(i,i'),R}$ respectively. The function $\mathsf{e}_{(i,i')|m}$ determines an upper bound to the size of any solution containing the pair $(i, i')$ together with all pairs in the current solution $m$. This bound is obtained by observing that for any submatrix $\mathcal{L}_{(i,i'),R}$ the number of rows and columns corresponding to pairs that can used in any solution must be equal: "only one cell can be used per row and per column".

To account for $m$, the LCM is updated so that if there is a pair in $m$ that is incompatible with any pair $(j, j')$, the entry for $(j, j')$ in $\mathcal{L}_{(i,i')}$ is artificially set to $\emptyset$. In this case a row or column that previously had non-empty cells might become empty potentially reducing the number of pairs that can be added to the current solution. The largest number of pairs that can be used as part of a solution containing $(i, i')$ given the current solution $m$ is

$$\mathsf{e}_{(i,i')|m} = \sum_{R \subseteq \mathbf{R}} \min\left(\mathsf{rows}(\mathcal{L}_{(i,i'),R}), \mathsf{cols}(\mathcal{L}_{(i,i'),R})\right)$$

The heuristic evaluation $\mathsf{e}_{(i,i')|m}$ almost always overestimates the extensibility of the solution $m$ after including the pair $(i, i')$. As a result an algorithm using these values to explore the search space may quickly run into basins of local optima. For hill climbing algorithms this is generally problematic because such basins may be difficult to escape.

A second heuristic computable upon return from the *move* that added $(i, i')$ to $m$ gives early feedback to the searcher. This heuristic function, called $\mathsf{e}_m$, aggregates the effects of all the pairs in $m$ on the extensibility of $m$. Suppose that for fixed $i \in N$ the pairs $(i, i')$, $i' \in N'$ are arranged in increasing order of $\mathsf{e}_{(i,i')|m}$, breaking ties randomly. Then the last item in this order, termed $\mathsf{head}_i$ determines the largest *possible* solution containing input node $i$. For $k \in \mathbb{N}$, $k \leq |N|$ let

$$\mathsf{count}_k(m) = \left|\{i \in N \mid k \leq \mathsf{e}_{\mathsf{head}_i|m}\}\right| .$$

Then $\mathsf{e}_m$ is given by

$$\mathsf{e}_m = \max_{\mathsf{count}_k(m) \geq k} k .$$

The presentation of the two heuristic evaluation functions that we gave in [3] did not suggest any approach for implementing them. In addition, from the description above a direct implementation would require $O(|N \times N'|^2)$ space and $O(|N \times N'|^3)$ computation time which is much more than what is strictly needed. In Section 4 we present a more efficient way to maintain the LCMs.

## 3 Learning Search Paths

The heuristics $\mathsf{e}_{(\cdot,\cdot)|m}$ and $\mathsf{e}_m$ do not interact in a dynamic way. Move selection is based on $\mathsf{e}_{(\cdot,\cdot)|m}$ because this is the information obtainable upfront. After a move is executed $\mathsf{e}_m$ provides feedback from the search space informing the searcher roughly whether the move was overvalued. Unfortunately, in order to maintain a consistent view of the search state, that feedback cannot be directly used to update $\mathsf{e}_{(\cdot,\cdot)|m}$. That is, in the absence of any manipulation of the move selection process, if the current solution were visited many times, the probability with which a particular move is executed in this state would remain unchanged. In this section we introduce a surrogate evaluation function to be used for exploration in the search process instead of $\mathsf{e}_{(\cdot,\cdot)|m}$. Because the surrogate does not directly depend on the global state, it can be updated with feedback from $\mathsf{e}_m$.

## 3.1 Reinforcement Learning

Those familiar with RL may have already spotted how the above description can be transformed into an RL model. In general RL has to do with learning by reinforcement [22, Chap. 3]; given a set $\mathcal{S}$ of states of the world, for each $s \in \mathcal{S}$, a set $\mathcal{A}_s$ of actions that an agent can take in state $s$, and for each legal state-action pair $(s, a)$ a set of observations, called *rewards*, that the agent makes upon executing an action $a$ in state $s$, the RL problem is the problem of maximizing expected returns of the form

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_{t+1}, a_{t+1}\right] \tag{1}$$

where $a_t$ is the action taken in state $s_t$, $r_{t+1}$ is the reward observed after performing action $a_t$ in state $s_t$, and $\gamma$ is the discount rate indicating the extent to which future rewards are taken into account. $\mathbb{E}$ returns the expected value of the expression within the square brackets. Actions are chosen according to a policy $\pi$ which can be understood as specifying transition probabilities between states. The function $Q_\pi(s_t, a_t)$ which gives the expected return at time $t$ when following policy $\pi$ is called an action value function for the policy $\pi$. In RL one often seeks to find the optimal value function for the optimal policy [22]. In our case we only seek to adjust the values so that any subsequent visit to a state-action pair takes into account previous visits in order to encourage moves that are likely to lead to an optimal solution.

Many algorithms have been designed to solve different variants of RL. The SARSA algorithm for on-policy reinforcement learning updates the state-action value function at iteration $t$ with the temporal difference (TD) error computed at iteration $t+1$ which depends on the action at that time:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{2}$$

The important fact for our work is that because SARSA is on-policy, we can use the same values that we are updating for exploration. For our search problem the learning agent is the searcher, the states are individual solutions, and the actions are the DROP or ADD moves. We will denote by $a_t(p)$ the action taken at iteration $t$ involving pair $p = (i, i')$ where $a_t \in \{$ADD, DROP$\}$.

**Value function initialization.** State-action values at state $m$ can be optimistically initialised to $\mathsf{e}_{p|m}$ for move ADD$(p)$ and $\mathsf{e}_{m-\{p\}}$ for move DROP$(p)$. In this initialization strategy $\mathsf{e}_{m_t-\{p\}}$ is only readily available if $a_{t-1} =$ ADD$(p)$. For any other pair $p' \in m_t$ the move DROP$(p')$ would have to be executed in order to obtain $\mathsf{e}_{m_t-\{p'\}}$. As shown in Section 2 this is expensive.

We obtain a much cheaper approximation for the initial values for DROP moves by introducing LCM$(m)$, an aggregate LCM for the current match $m$, and for each pair $p \in m$, LCM$^c(p)$, the complement LCM for $p$ with respect $m$. LCM$(m)$ is given by

$$\text{LCM}(m) = \bigwedge_{p \in m} \text{LCM}(p) \tag{3}$$

For a pair $p' \notin m$ we will say that $\mathsf{oneInconsistent}(p') = p \in m$ if the entry corresponding to $p'$ is empty in LCM$(m)$ and non-empty when LCM$(m - \{p\})$. We say $p'$ is profitably usable if any of the following hold

1. the row containing $p'$ in LCM$(m)$ is empty and $\mathsf{rows}(\text{LCM}(m)) < \mathsf{cols}(\text{LCM}(m))$, or

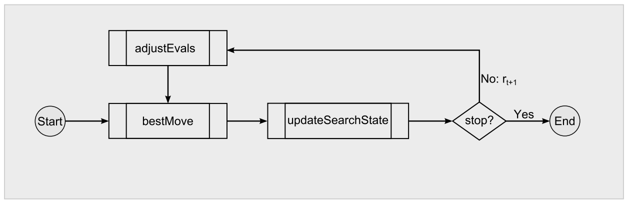


Figure 2: Main algorithm overview

2. the column containing $p'$ in $\mathrm{LCM}(m)$ is empty and $\mathsf{cols}(\mathrm{LCM}(m)) < \mathsf{rows}(\mathrm{LCM}(m))$

We then define $\mathrm{LCM}^c(p)$ as an LCM that has as non-empty entries cells corresponding to profitably usable pairs $p'$ such that $\mathsf{oneInconsistent}(p') = p$. The new LCMs are used to compute the potential $\rho(p)$ of $p$ which is the amount by which $\min\{\mathsf{rows}(\mathrm{LCM}(m)), \mathsf{cols}(\mathrm{LCM}(m))\}$ would increase if $p$ were to be removed from $m$. For the sake of readability let us denote $\mathsf{rows}(\mathrm{LCM}(m))$ by $\mathsf{rows}_m$, $\mathsf{rows}(\mathrm{LCM}^c(p))$ by $\mathsf{rows}^c_p$ and likewise for $\mathsf{cols}$. Then we have

$$\rho(p) = \min\{\mathsf{rows}^c_p + \mathsf{rows}_m, \mathsf{cols}^c_p + \mathsf{cols}_m\} - \min\{\mathsf{rows}_m, \mathsf{cols}_m\} \tag{4}$$

Thus the initial value for move $\mathrm{DROP}(p)$ in state $m_t$ is set to be

$$Q(m_t, \mathrm{DROP}(p)) = e_{m_t} + \rho(p) \tag{5}$$

Equation 5 is applied to every member of every solution at the first encounter - i.e. state-action values for DROP moves are only initialised after the state is visited.

**Reward function.** Let $\mathsf{e}_\emptyset$ be the heuristic evaluation at the empty solution. Then the basic reward at iteration $t$ is computed by

$$r_t = e_{m_t} - Q(m_t, a_t) - 1 + \frac{1}{1 + \mathsf{e}_\emptyset - |m_t|} \tag{6}$$

## 3.2 Searching with the RL based heuristic

In our implementation the RL update is embedded into our prohibition based metaheuristic algorithm as shown in the flowchart in Figure 2. Initially the procedure $\mathsf{bestMove}$ picks a pair $p$ to add to the current solution $m$ based on $Q(m, \mathrm{ADD}(p))$. Then after executing the move all data structures are updated in $\mathsf{updateSearchState}$ if the stopping conditions have not been met the procedure $\mathsf{updateEvals}$ which performs the learning updates is called with the reward value computed as above. The stopping conditions are that either a solution of size $\mathsf{e}_\emptyset$ has been found or a preset number of iterations of this main loop have been executed.

The implementation described above does not, however, account for the curse of curse of dimensionality. That is, the size of the state-action space prohibits us from storing the value function directly [17]. Convergence of the Q-values is guaranteed only if each state-action pair are visited an infinite number of times [22, Chap. 6]. Only visited state-action pairs will have their values updated which, in general, means that a large number of state-action pairs will have to be visited multiple times before an optimal ordering of Q-values that drives the search to the optimal solution is found.

To alleviate the memory challenge and to accelerate learning we introduce a method for approximating Q-values for moves at a solution using the Q-values of those moves at selected prototype solutions. Our approximations are based values attached to individual pairs in a solution.

**Prototypes and value function approximation.** For state value functions we may assume the pairs at a specific state (i.e. solution) contribute equally to the value of that state since a solution is completely determined by the set of pairs it contains. So a move $a(p)$ at state $m$, can be evaluated with respect to the pairs $p' \in m$. Let

$$Q_{p'}(m, a(p)) = \frac{Q(m, a(p))}{|m|}$$

then

$$Q(m, a(p)) = \sum_{p' \in m} Q_{p'}(m, a(p))$$

and for any two pairs $p', p'' \in m$ we have $Q_{p'}(m, a(p)) = Q_{p''}(m, a(p))$. The pairs $p'$ and $p''$ are *witnesses* of $p$'s moves at $m$. The SARSA update is then performed locally on the $Q$-value component at each pair. That is, for the sequence $m \xrightarrow{a} m' \xrightarrow{a'} m''$ we apply the update

$$Q(m,a) \leftarrow \sum_{p' \in m} \left[ Q_{p'}(m,a) + \alpha \left( \frac{r}{|m|} + \gamma Q_{p'}(m', a') - Q_{p'}(m,a) \right) \right] + \alpha\gamma Q_p(m', a') \tag{7}$$

where $arg(a) = p$ is the pair being moved. The term $Q_p(m', a')$ is 0 if $a =$DROP$(p)$ and it equals $\frac{Q(m',a')}{|m|+1}$ if $a =$ADD$(p)$.

Let $S$ be a subset of the state space. Updates for state $m$ can be backed up to each state $s \in S$ by updating the value at $s$ in the direction of the Temporal Difference (TD) error at $m$ by the proportion of $s$ that is in $m$:

$$\bar{Q}_{p'}(m,a) = \alpha \left[ \frac{r}{|m|} + \gamma \left( \hat{Q}_{p'}(m', a') + \frac{\hat{Q}_p(m', a')}{|m|} \right) - \hat{Q}_{p'}(m,a) \right] \tag{8}$$

where $\hat{Q}(m,a)$ is the approximation of $Q(m,a)$ computed from the *prototypes* in $S$. The update at pair $p'$ of $s$ is then obtained by scaling the TD error from $m$ to $s$ and applying equation (2)

$$Q_{p'}(s, a_t) \leftarrow Q_{p'}(s, a_t) + \frac{|m|}{|s|} \bar{Q}_{p'}(m,a) \tag{9}$$

Combining equations (7) and (9) the full update to prototype $s$ is

$$Q(s, a_t) \leftarrow Q(s, a_t) + \frac{|m \cap s|}{|s|} \bar{Q}(m_t, a_t) \tag{10}$$

Notice that no reference to the individual pairs is used in the update. To read out the approximate value for action $a_t(p)$ at state $m$, the individual values at each prototype containing each pair $p' \in m$ are scaled, normalized to the number of prototypes containing $p'$, and then aggregated. Let $S(p')$ be the set of prototypes containing $p'$. Then

$$\hat{Q}(m, a_t(p)) = \sum_{p' \in m} \left( \frac{1}{|S(p')|} \sum_{s \in S(p')} \frac{Q(s, a_t(p))}{|m|} \right) \tag{11}$$

The scaling of values between prototypes and solutions is necessary to account for differences in the sizes of the solutions. Next we describe a simple strategy for selecting which states serve as prototypes.

**Maintaining prototypes.** Prototypes are states that are chosen to represent other states that are similar to them in some way. For QCN matching, we would like the prototypes to coincide with legal solutions of the problem. Meaning that we should create prototypes only from solutions that have been visited either directly or by proxy (e.g. a neighbour of a visited solution can be determined by simply checking the moves out of that solution).

Prototypes for solutions encountered during search are determined using the idea of a set cover, a distance function, and a single constraint. The distance function we use is based on set intersection and union. Let $m, m'$ be two solutions. The distance between $m$ and $m'$ is given by the Jaccard distance [13]

$$d(m, m') = 1 - \frac{|m \cap m'|}{|m \cup m'|} .$$

Given a set $X$, a collection $S$ of subsets of $X$ is said to cover $X$ if $\cup S = X$. We remove the requirement that $S$ need be subsets of $X$ and demand only that $\cup S \supseteq X$. We consider a minimal cover $S' \subseteq S$ for $X$ to be a cover for $X$ that is minimal in the sense that given any other cover $S'' \subseteq S$ for $X$, $|S'| \leq |S''|$.

A solution, $m$, in our algorithm must have two sets of prototypes denoted $S_m$ and $S'_m$ satisfying

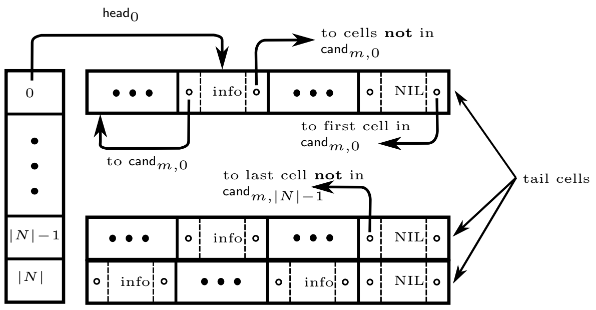
1. $S_m$ is a minimal cover for $m$ and $\forall s \in S_m$, $d(s, m) \geq \mathsf{minDist}$, and
2. $\forall s' \in S'_m$, $s' \backslash m \neq \emptyset$ and $d(s', m) < \mathsf{minDist}$ and $|S'_m| = |S_m|$.

$\mathsf{minDist}$ is a function of $m$ decided at initialization that returns a threshold in the interval $(0, 1)$. The sets $S_m$ and $S'_m$ are generated dynamically for each solution $m$ and may be regenerated since excess prototypes at a solution may be removed randomly during the course of search. If $|S'_m| < |S_m|$ then $|S_m| - |S'_m|$ prototypes are created by first taking prototypes $s$ satisfying $d(s', m) < \mathsf{minDist}$ but not $s' \backslash m \neq \emptyset$ (or vice-versa) and extending them by a pair consistent with but not contained in $m$ (or in the alternate case adding pairs in $m$ to $s'$ until $d(s', m) < \mathsf{minDist}$). If no such prototypes exist then a copy of $m$ is taken from which are removed the maximum number of pairs that can be removed without violating the second condition before the resulting solution is extended by a pair not in $m$. This is repeated until $|S'_m| = |S_m|$.

For $S_m$, we first check if $S \backslash S_m \neq \emptyset$. If so then solutions from $S \backslash S_m$ are taken one at a time until either condition 1 is met or $S \backslash S_m$ is exhausted. With each solution taken from $S \backslash S_m$ that intersects $m$ a new solution is created by removing from the former pairs in $m$ until the result $s$ satisfies the distance constraint $d(s, m) \geq \mathsf{minDist}$. When no more solutions in $S \backslash S_m$ intersect $m$, then $m$ is used instead of $s' \in S \backslash S_m$. These steps are repeated until a cover is achieved.

Prototypes are deleted in two situations. If $|S'_m| > |S_m|$ then $|S'_m| - |S_m|$ prototypes are deleted from $S'_m$. When the memory is full prototypes are randomly deleted. The size of the memory is set at initialization. In implementation the sets $|S_m|$ and $|S'_m|$ are updated incrementally since only prototypes containing the pair $p'$ which was moved in the transition from solution $m_t$ to $m_{t+1}$ may introduce inconsistencies.

**Summary.** The SARSA based algorithm presented in this section encapsulates the main aspects of our solution to the QCN matching problem (QMP). The generalization accelerates learning but a small computation cost. In the next section we give details of how the overall computation is made more feasible by employing efficient data structures for the LCMs.



head$_0$
to cells **not** in cand$_{m,0}$
0
info
NIL
to first cell in cand$_{m,0}$
to cand$_{m,0}$
to last cell **not** in cand$_{m,|N|-1}$
tail cells
$|N|-1$
$|N|$

Figure 3: The indicator-set array

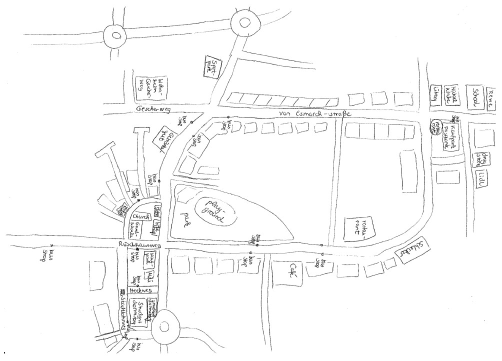


Figure 4: Sketch map corresponding to results in Figure 5

## 4 Efficient Implementation of LCMs

To compute $\mathsf{e}_m$ for each $m$ may require $O(|N \times N'|^3)$ iterations of an iterative update algorithm such as path-consistency as done in [23]. But considering the large number of possible moves that must be considered at each iteration of the main algorithm it is necessary to find more efficient ways to update the search state.

**Candidate lists.** We represent the pairs available for adding to the current match by a list of subsets of $N \times N'$ where each node $i \in N$ indexes exactly one subset in the list. We define these subsets by

$$\mathsf{cand}_{m,i} = \{(i, i') \mid \forall (j, j') \in m, l(i, j) \cap l'(i', j') \neq \emptyset\}$$

and build on the *indicator set* data structure introduced by Battiti and Protasi in [1] to come up with an efficient data structure for maintaining them. An indicator set, is an array of records formed by three components: a pointer to a previous entry, some application data called the info component, and a pointer to a next entry. The previous and next pointers are integer indexes into the array and impose an ordering on the cells that is possibly different from the natural ordering imposed by the array indexing. The advantage of such a structure is that the info in each cell can be accessed in constant time provided we know which cell it resides in. For each $i$, there is an additional initial record pointer that identifies the initial record in the linked list.

**Indicator set array.** An *indicator set array*, as the name suggests, is an array of indicator sets indexed by some attribute of the data stored in the set (see Figure 3). In our case each of the sets $\mathsf{cand}_{m,i}$ is implemented as one portion of an indicator set and the indicator set array is indexed by the input nodes $i \in N$. The other portion of the indicator set (the portion that *does not* contain $\mathsf{cand}_{m,i}$) consists of records whose info components contain pairs $(i, j)$ that are *not* compatible with at least one pair in the current solution.

The $i'^{th}$ cell of the indicator set at index $i$ in the indicator set array corresponds to the pair $(i, i')$. For each $i$ and each match $m$, there is a cell $\mathsf{head}_i$ such that for all $(i, i') \in \mathsf{cand}_{m,i}$, $\mathsf{e}_{(i,i')|m} \leq \mathsf{e}_{\mathsf{head}_i|m}$. In particular, $\mathsf{head}_i \in \mathsf{cand}_{m,i}$.

The records of an indicator set whose pairs belong to $\mathsf{cand}_{m,i}$ are ordered by their values of $\mathsf{e}_{(i,i')|m}$ with $\mathsf{head}_i$ always taking the last place. When a pair is moved out of $\mathsf{cand}_{m,i}$ it is placed immediately after $\mathsf{head}_i$. This operation involves six value assignments to update the next and previous pointers of the pair being moved all pointers pointing to it before and after the move. To move a pair into $\mathsf{cand}_{m,i}$ one simply searches $\mathsf{cand}_{m,i}$ for the appropriate position to install the pair and updates the pointers as above. This takes at most $|N'|$ comparisons plus the six pointer updates.

During an add move all pairs $(i, i')$ that are not compatible with the pair just added to $m$ are moved out of $\mathsf{cand}_{m,i}$. If the move being executed is a drop move, steps that reverse the steps taken during add moves are performed. That is, pairs $(i, i')$ such that $(i, i') \notin \mathsf{cand}_{m,i}$ and $(i, i')$ is not compatible with the pair that was just removed from $m$ (but is compatible with all pairs remaining in $m$) are placed into $\mathsf{cand}_{m,i}$.

Updating $\mathsf{e}_{(i,i')|m}$ implies updating the dimensions of the corresponding LCM. The dimension pairs (height and width) for the entire set of LCMs are implemented using a pair of 2-dimensional arrays:

$$\mathsf{labelRows}_{(i,i'),R} = |\{j \in N \mid \exists j' \in N',\ (j,j') \in \mathcal{L}_{(i,i'),R}\}| \ ,$$
$$\mathsf{labelCols}_{(i,i'),R} = |\{j' \in N' \mid \exists j \in N, (j,j') \in \mathcal{L}_{(i,i'),R}\}| \ .$$

We update the two arrays above by counting for a pair $(i, i')$ and each input node $j$ the number of pairs $(j, j')$ in $\mathsf{cand}_{m,j}$ that are compatible with $(i, i')$ and for each $\iota' \in N'$ the number of nodes $\iota \in N$ such that $(\iota, \iota') \in \mathsf{cand}_{m,\iota}$ and $(\iota, \iota')$ is compatible with $(i, i')$. If moving a pair $(j, j')$ out of $\mathsf{cand}_{m,j}$ results in the smaller of the values $\mathsf{labelRows}_{(i,i'),R}$ or $\mathsf{labelCols}_{(i,i'),R}$ being decremented, $\mathsf{e}_{(i,i')|m}$ is also decremented. Here $R$ is the intersection of labels $l(i, j) \cap l'(i', j')$.

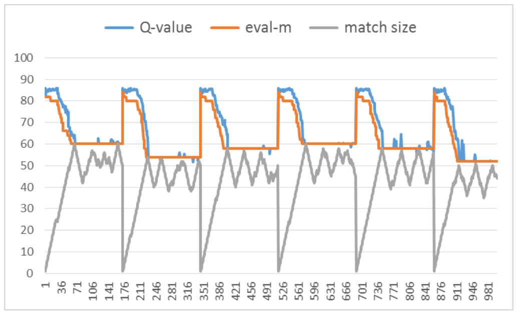
Updating $\mathsf{e}_m$ involves maintaining a threshold $\mathsf{evalCounter}$ which is the number of nodes $i \in N$ for which $\mathsf{e}_{\mathrm{head}_i|m} > \mathsf{e}_m$. The essence of this number is that if it is itself greater than $\mathsf{e}_m$ then the current match $m$ is under-valued and that its value must be incremented. To decrement $\mathsf{e}_m$ we must know when the number of nodes $i \in N$ such that $\mathsf{e}_m \leq \mathsf{e}_{\mathrm{head}_i|m}$ falls below the value $\mathsf{e}_m$ itself. This would mean that no match from that point on could be extended to a match containing at least $\mathsf{e}_m$ pairs. So the global evaluation must be decremented. The record of these numbers is stored in an array of size $N$. The (in/de)crementing is done in steps of size 1 because each time *at most* one item at the head of a candidate list can change. The following results derive directly from the foregoing discussion and the tacit assumption that the number of labels used $|l(\mathcal{N} \uplus \mathcal{N}')| = O(1)$.

**Theorem 1.** *In each iteration of local search, the state space including the Local Compatibility Matrices consume $O(|N| \cdot |N'|)$ space and require $O(|N|^2 \cdot |N'|^2)$ steps to update.*

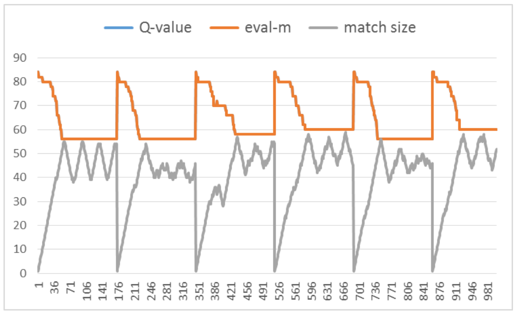
The actual number of computation steps per iteration is in fact much smaller since the candidate lists and the LCMs are most of the time below their initial sizes. In the next section we show representative results of an evaluation of the present algorithm against a version that does not incorporate learning. We also compare our results with results obtained in our previous work [3].

## 5 Evaluation

We performed an evaluation of our matching algorithm with and without learning on the sketch-to-metric map alignment problem. The experiment was run on a desktop workstation computer with an Intel$^{\mathrm{TM}}$Core i5 processor, (2.60 GHz) and 12 GB RAM. The operating system was Windows 7, 64-bit and algorithms are implemented in Java and executed on the Oracle Java JVM 1.7.0_71b14.



(a)



(b)

Figure 5: A comparison of the first 1000 iteration of two the tabu search based metaheuristic with periodic restarts:(a) with learning (b) without learning.

Here we discuss a summary of the results of our evaluation for the example sketch map shown in Figure 4. Figure 5 shows the progress of one run each for the algorithm with learning and one without learning. Looking at the two graphs one observes that in graph (b) there is a separation between the curve representing the current match size and the predicted best match from that point on (as given by $e_m$). In the graph above however the match size always reaches its optimal value in regions of the search space where the heuristic evaluation has stabilized. In all experiments, the version with learning performed better than its counterpart which did not include a learning component.

## 6 Conclusion

Learning is powerful way to enhance search algorithms for computationally hard problems. We have outlined an approach that uses an online RL component to dynamically update heuristic evaluations on solutions encountered along search paths. The interesting outcome is that matching performance actually improves. Following a standard approach we have been able to solve an example that we were unable to do with previous methods of [23], [16], or our previous work [4, 3].

Learning in our algorithm was implemented using SARSA with a novel generalization approach. SARSA is well known and extensively studied RL algorithm that has been shown to perform well with generalizations based on Kanerva's distributed sparse memories (SDM) [18]. Like [18] we use a single parameter minDist to control the number of prototypes that a particular state backs up its value to and generate these prototypes dynamically. But unlike that work we do not use an activation radius to select which prototypes a state is backed up to. All prototypes sharing a node with the current solution receive a portion of the signal emitted by the SARSA update at that state.

In this paper we were interested in the practical aspects of the QCN matching problem. The LCMs presented in Section 2 can handle non-atomic constraints which is one way of performing inexact matching for spatial configurations. In addition several spatial calculi could be used to describe the input data in the LCM framework.

The icing on the pudding is the ability to dynamically adapt the LCM structure which supports incorporating spatial aggregation and splitting into the matching process. This is a difficult problem considering that aggregations by themselves introduce another combinatorial explosion on top of the exponential search space of the basic QMP. In this regard, learning to associate potential aggregations with profitable regions of the search space might help mitigate the adverse effects of this exponential expansion of the search space. This is the ultimate challenge that we intend to overcome going forward.

## 7 Acknowledgments

The author would like to thank the two anonymous referees for their comments, which helped improve the paper. Many thanks go to Angela Schwering without whose support the preparation of this paper would not have been possible and to Carl Schultz and Sahib Jan for many discussions that helped shape the ideas presented here.

## References

[1] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[2] M. Chipofya. A qualitative reasoning approach for improving query results for sketch-based queries by topological analysis of spatial aggregation. Master's thesis, Universitat Jaume I, Universität Münster, Universidad Nova de Lisboa, February 2010.

[3] M. Chipofya, C. Schultz, and A. Schwering. A metaheuristic approach for efficient and effective sketch-to-metric map alignment. *International Journal of Geographical Information Science*, 30(2):405425, October 2015.

[4] M. Chipofya, A. Schwering, and T. Binor. Matching qualitative spatial scene descriptions á la tabu. In Félix Castro, Alexander Gelbukh, and Miguel González, editors, *Advances in Soft Computing and Its Applications*, volume 8266 of *Lecture Notes in Computer Science*, pages 388–402. Springer Berlin Heidelberg, 2013.

[5] M. Chipofya, A. Schwering, C. Schultz, E. Harason, and S. Jan. Left-right relations for qualitative representation and alignment of planar spatial networks. In O. Pichardo, O. Herrera, and G. Arroyo, editors, *Proceedings of the 14th Mexican International Conference on Artificial Intelligence, MICAI 2015*, Cuernavaca, Mexico, 2015. Springer International Publishing.

[6] A. G. Cohn, B. Bennett, J. Gooday, and N. M. Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *Geoinformatica*, 1:275–316, October 1997.

[7] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems*, volume 19 of *Neural Information Processing*, pages 313–320. MIT Press, 2007.

[8] G Edwards, G Ligozat, B Moulin, CM Gold, et al. A voronoi-based pivot representation of spatial concepts and its application to route descriptions expressed in natural language. In *Proceedings of Spatial Data Handling96*, 1996.

[9] M. Egenhofer. Spatial-query-by-sketch. In M. Burnett and W. Citrin, editors, *IEEE Symposium on Visual Languages*, pages 60–67, Boulder, CO, 1996.

[10] L. Frommberger and D. Wolter. Structural knowledge transfer by spatial abstraction for reinforcement learning agents. *Adaptive Behavior*, 2010.

[11] F. Glover. Tabu search – part 1. *ORSA Journal on computing*, 1(3):190–206, 1989.

[12] X. Jiang and H. Bunke. Graph matching. In P. Perner, editor, *Case-Based Reasoning on Images and Signals*, volume 73 of *Studies in Computational Intelligence*, pages 149–173. Springer, 2008.

[13] Michael Levandowsky and David Winter. Distance between sets. *Nature*, 234(5323):34–35, November 1971.

[14] G. Ligozat. Categorical methods in qualitative reasoning: The case for weak representations. In A. G. Cohn and D. M. Mark, editors, *Spatial Information Theory*, volume 3693 of *Lecture Notes in Computer Science*, pages 265–282. Springer Berlin Heidelberg, 2005.

[15] G. Ligozat and J. Renz. What is a qualitative calculus? a general framework. In C. Zhang, H. W. Guesgen, and W.-K. Yeap, editors, *PRICAI*, volume 3157 of *Lecture Notes in Computer Science*, pages 53–64. Springer, 2004.

[16] K. Nedas and M. Egenhofer. Spatial-scene similarity queries. *Transactions in GIS*, 12(6):661–681, 2008.

[17] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley and Sons, 2011.

[18] B. Ratitch and D. Precup. Sparse distributed memories for on-line value-based reinforcement learning. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004*, pages 347–358, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[19] J. Renz and B. Nebel. Qualitative spatial reasoning using constraint calculi. In M. Aiello, I. Pratt-Hartmann, and J. Van Benthem, editors, *Handbook of Spatial Logics*, chapter 4, pages 161–215. Springer, 2007.

[20] A. Schwering, J. Wang, M. Chipofya, S. Jan, R. Li, and K. Broelemann. Sketchmapia: Qualitative representations for the alignment of sketch and metric maps. *Spatial Cognition & Computation*, 14(3):220–254, 2014.

[21] M. Skubic, D. Anderson, S. Blisard, D. Perzanowski, and A. Schultz. Using a hand-drawn sketch to control a team of robots. *Autonomous Robots*, 22(4):399–410, 2007.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* The MIT press, Cambridge, Massachusetts, 1998.

[23] J. O. Wallgrün, D. Wolter, and K.-F. Richter. Qualitative matching of spatial information. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 300–309, New York, USA, 2010. ACM.