A Fast Mapper as a Foundation for Forthcoming Conceptual Blending Experiments

João Gonçalves, Pedro Martins, and Amílcar Cardoso

CISUC, Department of Informatics Engineering, University of Coimbra
{jcgonc,pjmm,amilcar}@dei.uc.pt

Abstract. Algorithms for finding analogies as mappings between pairs of concepts are fundamental to some implementations of Conceptual Blending (CB), a theory which has been suggested as explaining some cognitive processes behind the creativity phenomenon. When analogies are defined as sub-isomorphisms of semantic graphs, we find ourselves with a NP-complete problem. In this paper we propose and compare a new high performance stochastic mapper that efficiently handles semantic graphs containing millions of relations between concepts, while outputting in real-time analogy mappings ready for use by another algorithm, such as a computational system based on CB theory.

1 Introduction

In a paper titled *Analogy as the Core of Cognition* Hofstadter [14] stated that psychologists and cognitive scientists consider analogy as a sophisticated and cryptic mental tool used in problem-solving, especially artistic people [11]. Multiple authors [2,11,22] declare that analogy is a essential task in many creative processes. In our current main focus - Computational Creativity (CC) - researchers are typically concerned building algorithms for the creation of new ideas or the display of behaviours exhibiting creativity. From these we refer to a few which have influenced our work in [19,3,24,1,13]. Similar to other researchers in Computer Science we try to bring fruitful findings in cognitive science into implementing CC systems.

Simply stated, analogy is a method which relates ideas or pieces of thought. Analogy as a process can be easily observed when asking someone "*if* ***a*** *is to* ***b*** *then what is* ***c*** *to*?" Or the question "*the* ***composer*** *is to the* ***general*** *as the* ***drum*** *is to*?" [24] This question should develop in most people answers such as *cannon*, *bomb*, *tank*, etc. giving us a sense that analogy seems to be at the core of cognition [26]. Analogy emerges as a fundamental tool to associate the above two domains of knowledge - the composer and the general. It is not hard to think that this process can be taken further by combining (or *blending*) partially selected parts of information from the two domains into a new mental idea, for instance, that the composed is directing an orchestra of cannons and bombs emerging in a scenery of musical warfare... We are of the opinion that this text seems to possess some form of creativity [11]. In a nutshell, this is the idea behind

CB, a cognitive theory which has been successfully applied in CC systems [19,3] including international projects such as CoInvent [20] and ConCreTe [27]. CB requires a mapping of concepts - an analogy - to be effective and is elaborated in the next section.

The contribution described in this paper flows from the current work of our research group in developing a computational system which in the future is expected to exhibit creativity. We previously reported [9] a CB module (named the Blender) based on a evolutionary algorithm which requires as an input a semantic graph and one or more analogies in the form of sets of concept pairs. These mappings are typically either produced manually (to validate a system) or produced by an existing computational analogy system such as the ones outlined in [6]. However, as far as we know we have not found an algorithm fast enough to extract in real-time mappings from giant non trivial semantic graphs available in the web, such as the Never-Ending Language Learning (NELL) project [17] and ConceptNet [21]. Our vision is that in a fully automated process the analogies should follow the frames pertinent to the blend (an instance of the blending space) being elaborated in the Blender module. In turn this requires a sort of feedback loop between the blender and the mapper (which creates suitable mappings) and therefore, a mapping module (the focus of this paper) capable of executing in real-time and concurrently with the blending module.

This paper is laid out as follows: we start with a short overview of CB and why it requires an analogy in the form of a mapping to work; it is followed by the description of a few computational analogy systems which were relevant for our work and CB in general; then we reveal our EEmapper and its inner workings; afterwards we compare it against an optimum mapper and jMapper and examine the results; finally, we outline further work to improve our mapper and conclude on our findings. Semantic graphs are given as relations in the form *(source,relation,target)*, an analogy is represented as a functional mapping between two sets of concepts and the size of a mapping corresponds to the number of one-to-one associations between concepts in the mapping set.

2 Conceptual Blending and the importance of analogies

Fauconnier and Turner [5] suggested CB as cognitive theory to explain processes of conceptual integration occurring in the human thought. Its potential to model mechanisms of concept invention has increasingly inspired research in CC in recent years [27,20] and also in our research group [19,9,3]. Initially, CB was proposed as a framework for conceptual metaphor theory, i.e., a cognitive explanation of the reasoning behind metaphors as a linguistic phenomenon. Later [8] it was adapted to include the projection of one information domain to another as a result of an analogy process with a partial recombination of concepts and relations characteristic of the blending process. The projection of concepts or analogy corresponds to a one to one mapping between concepts of different domains being one the source and the other the target. In graph theory, the analogy (or mapping) is defined as a structural alignment of concepts from different

domains of knowledge and can be seen as a graph isomorphism between two semantic graphs or regions of a larger semantic graph. This way, CB includes the usage of analogy to blend elements from different thoughts and is equipped to explain the synthesis of new ideas and the everyday language [8].

An essential ingredient in the CB theory is the *mental space*, a partial and temporary structure of knowledge assembled for purposes of thought and action [5]. The CB process takes two *input spaces* and looks for a partial *mapping* between elements of both spaces that may be perceived as similar or analogous in some respect. A third mental space, called *generic*, encapsulates the conceptual structure shared by the input spaces, providing guidance to the next step of the process, where elements from each of the input spaces are *selectively projected* into a new mental space, called the *Blend Space*. Further stages of the process elaborate and complete the blend.

In the CC and Analogy fields the input spaces have been typically given in the form of semantic graphs, that is, graphs with directed edges representing relations between concepts. Mappings represent analogies and are defined as sets of ordered pairs of concepts, each concept usually coming from a different input space than the other concept in the pair. Mappings represent a one-to-one correspondence between concepts of different regions of the semantic space [18]. In this paper we term the computational systems which generate mappings as *mappers* and describe the ones we find crucial for our work below.

2.1 Structural Mapping Theory

In [7] Gentner conceived her Structure Mapping Theory (SMT) stating that in order to establish an analogy, two domains of knowledge defined by interconnected relations between concepts, are matched from one domain to the other. This matching is on its essence a structural alignment, or mapping, of one to one correspondences between concepts from both domains. First and foremost, particular crucial associations between the concepts from both domains are what will identify the analogy. However, more interesting results could materialize if the associations are fabricated using higher order relations, that is, relations between relations or some sort of abstract matching [8]. SMT gave rise to a robust algorithm - Structural Mapping Engine (SME) [4] - which probably is the most significant and earlier work in Computational Analogy [6].

2.2 Sapper

Sapper[25] is one of the first mappers offered as an alternative to Gentner's SME. Sapper was initially described as a model of memory for metaphor comprehension. Since then it has also been used as a dedicated mapping engine by itself [24] or as a foundation for developing further mappers [19,12]. Given two inputs in the form of semantic graphs representing the *Tenor* domain and the *Vehicle* domain (the components of a metaphor), Sapper lays out what the authors label as dormant bridges - one-to-one associations between concepts.

Sapper works as a spreading activation mechanism in the given input semantic graphs and thus, it is in a sense a hybrid algorithm integrating principles from symbolic computation with connectionist philosophies. Sapper works in batches of two phases exchanging information between a structural inference phase (mostly the Triangle and Square rules) and the opportunistic activation phase. Is in the second phase where the limits of the mapping are defined by checking how important the nearby pairs of concepts are.

The Triangle rule lays out a dormant bridge (equivalent to a association of two concepts) whenever those two concepts share a common concept with the same relation. For instance, if *dog,isa,animal* and *cat,isa,animal* the shared concept is *animal* through the *isa* relation. In this example Sapper associates both dog and cat concepts in the first phase. The Square rule builds on previously laid associations of concepts and if these also share the same relation with a third and a forth concepts, Sapper associates the latter two. Again, as an example if *dog* and *cat* are associated and the semantic network contains the two relations *cat,atlocation,crib* and *dog,atlocation,doghouse*, Sapper lays down a new association, in this case between the concepts *crib* and *doghouse*. When Sapper is complete it returns the largest mapping (in number of associations) containing the dormant bridges activated during its execution.

2.3 jMapper

In [19] Pereira developed in Prolog a mapper which found analogy mappings between concepts from two input spaces, using a structural alignment algorithm based on Sapper [24]. In [12] Pereira's mapper was re-implemented in Java with gains in efficiency and scalability, although maintaining the original idea. The authors state that jMapper reduces the search space and ranks the pairs of concept candidates in terms of potential similarity. This similarity is based on the number nearby relations shared between each concept and their nearby concepts. The mapper allows a similarity threshold to be set that avoids the exploration of portions of the search space. For instance, if a low threshold is set a region of the mapping which associates animals with plants could stop if their only relation in common is that they are a form of life. As such, jMapper prefers to explore concepts from the semantic graph that have more in common.

As stated above, jMapper has its roots on Sapper and looks for pairs of concepts that share the same relation to a third concept (the Triangle rule). From then on, it applies the Square rule to look for 1-to-1 correspondences. In the end, jMapper returns the largest mappings.

2.4 Optimum Mapper

This mapper was previously developed in our research group to investigate the complexity and feasibility of finding mappings in various semantic networks of diverse sizes [3]. As the name indicates, the algorithm is exhaustive and optimal, as it will create all sets of possible mappings in order to find the largest achievable analogy, that is, the mapping set with the greatest number of concept pairs. As

the algorithm serves as a theoretical basis for the EEmapper we give a short summary next.

The algorithm begins in a root pair composed of two distinct concepts taken from the input spaces. Both concepts are not required to be related and thus contrasting the Triangle Rule in Sapper/jMapper. Then, the execution is performed in two stages. In the first stage, the algorithm finds a structural isomorphism in the global input space (combination of both input spaces), extracting two isomorphic sub-graphs. This isomorphism is edge based and reflects the same sequence of relations in the sub-graphs. We term the two input spaces *left* and *right*. Starting at the root pair, the isomorphic sub-graphs are extracted from the input spaces by executing two synchronized expansions of nearby concepts at increasingly depths, one from the left and the other from the right concepts defining the root pair. The expansion is done recursively in the form of a hybrid between a depth first expansion and a breadth first expansion, one expansion dedicated for the left sub-graph and the other to the right sub-graph. The left and right isomorphic sub-graphs define a mapping composed of a unique set of ordered pairs of concepts. Each concept of a pair comes from its respective left or right isomorphism and thus, from one of the input spaces. Any concept belonging to a pair is excluded from further expansions and future pairs.

While expanding, the algorithm stores additional associations between each matched relations and the corresponding concept which was reached through that relation. In reality, what is likely to happen is to occur a multitude of isomorphisms. In that case the algorithm will store various concept pairs relating any given concept to multiple matching concepts, as long as the same concepts where reached from a previous concept with the same relation. This is the basis to find an edge/relation based sub-graph isomorphism. The last stage corresponds to iterating all the isomorphisms found in the first stage and extracting the largest mapping in terms of concept pairs. It may happen that there are multiple mappings with the same size and in either case, all the equally largest mappings are outputted as analogies.

3 EEMapper

Our proposed algorithm for extracting mappings from semantic graphs is titled EEmapper and is based on evolutionary principles. Although there is quite a number of mappers in the literature, including the ones explained before, none is fully capable of handling semantic graphs in the order of millions of relations. Either the mappers do not halt at all or consume a vast amount of resources (memory included) and trigger software exceptions due to the combinatory explosion of possibilities they have to explore to output a mapping. There are also other problems such as the wait for a usable mapping. When a CB system depends on a mapping to do its elaboration and the mapping is not available, or if the mapping should change in a small part to allow the CB task to improve its results it is clear that the mapping engine must be fast enough to nourish the dependent tasks. In this case we have an optimisation issue inside of another

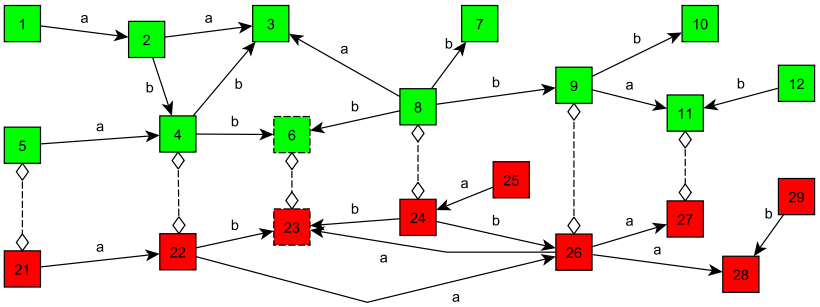


Fig. 1: Example of a mapping (analogy) between two domains of knowledge (green and red) of a larger semantic graph. Associations are shown with vertical dashed lines between concepts. Best viewed in colour.

optimisation problem and therefore the combined complexity must be lowered if we expect results in a useful time. Hence the purpose of this paper, a somewhat embryonic but highly fast mapper to find the largest mapping (in number of concept pairs - one to one associations of concepts). Although currently missing, in the future we expect this search to include a form of semantic quality or even of usefulness for a CB framework.

It is well known in Computer Science that finding the perfect answer to a problem (in our case the ideal mapping) may be impossible to reach. The only alternative then is to find an answer "good enough" to our problem and that is when we turn ourselves to stochastic algorithms. Both our CB framework [9] and our Domain Spotter [10] are based on a High Performance Genetic Algorithm (GA) and because of this, we have a good foundation for building the EEmapper. Our GA is prepared to handle multiple threads in parallel while minimising memory usage. The GA runs in multiple batches of three phases in parallel corresponding each batch to an epoch in the evolution of the population. The three phases are the population mutation and crossover (genetic operations phase), population selection for the next generation (k tournament selection) and fitness evaluation. Our EEmapper currently does not implement crossover and has a simple but fast mutation. Hence we consider it a system with evolutionary principles and not a fully GA, but this is expected to change in the future. Given the scope of this paper we do not describe the inner workings of a typical GA but only the required operations to implement the EEmapper. The mapper is founded on a partial but faster version of the isomorphism described in the optimum mapper.

Each chromosome is defined by what we name a *root pair* of two different concepts (vertices 6 and 23 in Fig. 1 representing a one to one association and the building blocks of the emerging mapping. Any association of concepts is possible, given the stochastic nature of the algorithm. In each epoch, EEmapper evolves hundreds or thousands of chromosomes, each representing a mapping. Hence the system not only outputs the best result but multiple mappings similar

or resulting from a fluctuating population of chromosomes. Excluding speed optimizations in the algorithm (such as only running the fitness function whenever a chromosome changes) the fitness function reflects the execution of the stochastic sub-isomorphism finding algorithm, naturally applied to every chromosome of each generation. As in the Optimum Mapper, the score for each chromosome (and related mapping) is the number of concept pairs in the mapping. For instance, if a chromosome has stored in its genetic material the mapping shown in Fig. 1 its fitness score would be 6.

When the GA first starts, the root pairs are randomly generated, i.e., the mapper starts from an association such as *(dog,cat)* or from *(rock,light)*. This decision was taken so that the system could find mappings between disconnected domains even if at first sight they appear completely unrelated, as this may discover extraordinary findings [15]. The partial isomorphism matching algorithm executes whenever a chromosome changes (because of a genetic operation) or is reset (build from scratch or in the GA's first population). The algorithm randomly chooses the same path of relations (labels such as *isa*, *partof*, *atlocation*) in the left and right input spaces shown in the same Fig. This is equivalent to executing the full left and right expansion in the optimum mapper for later only choosing a single random mapping from that expansion, that is, one path from the root to a leaf in the deep first expansion of the semantic tree. Therefore, a mapping depends on the initial root pair and the (random) sequence of branches (associations of relations and thus concepts) generated by the Random Number Generator. In a sense, the defining of a root pair is equivalent to Sapper's Triangle rule and the mutation's random walk to Sapper's Square rule. However, contrary to Sapper, the EEmapper works without problems in disconnected input spaces. With the above described stochastic process executing in the mutation operator the search space is cut in a tiny fraction of the other mappers with the expected outcome of lowering the probability of finding the largest mapping.

The mutation mechanism does two types of operations. First, it randomly changes one or both of the concepts in its root pair to nearby connected concepts. This mechanism has a progressively smaller chance of randomly mutating a concept to a more distant neighbour. As an example, if a root pair is *dog,cat* a possible mutation is *canine,cat* or in Fig. 1 changing the root pair *6,23* to *6,24*. Second, the mutation re-executes a new partial isomorphism match from the existing or mutated root pair in the hope of shuffling the resulting mapping.

4 Comparative evaluation

As an initial validation of our new mapper, we compared it against an optimal mapper we developed previously in our research group [3] and against an efficient implementation of Sapper - jMapper [12]. This comparison is as of this document done ingenuously in the form of the number of concept pairs present in the resulting mappings and naturally, the time required to obtain those mappings. We admit we are neglecting the semantic features of the mappings but as expressed

in the Section 6, further study is expected in this aspect. The experiments were done with the following criteria in mind:

- for what size of the input spaces do the optimum mapper and jMapper fail to converge?
- for the time optimum mapper and jMapper took to execute what are the largest mappings obtained by EEmapper?
- if possible, how long does EEmapper require to reach similar results to the other mappers?

jMapper requires two input spaces with one requirement - the existence of common concepts in both input spaces to be used in the Triangle rule [24]. On the other hand, both our EEmapper and optimum mapper do not have this requirement. To use jMapper we decided to use our semantic graph splitter, titled Domain Spotter [10] to partition ConceptNet in various pairs of input spaces to be used in the experiments. The Spotter is based on the theory of Bisociation and extracts from a given semantic graph two apparently unrelated domains - the input spaces - ideally connected only through a single term [15] - the bridge node which is always present in both extracted input spaces.

The Domain Spotter runs as a Genetic Algorithm and aims to maximise the number of concepts present in both generated input spaces while minimising their intersection. It includes a penalisation parameter τ to control how hard is the separation (in concepts) of both input spaces. The higher the τ is, the higher the amount of concepts in the intersection between both extracted input spaces. In this case, jMapper will have a high amount of concepts available to apply its Triangle rule. The Spotter contains various parameters for fine-tuning the movement of the bridge vertex in the search space, with γ defining the nearby range where the bridge concept can move to.

To evaluate our EEmapper against the optimum mapper and jMapper we turned to ConceptNet v5, a known semantic network built from information collected by the Open Mind Common Sense project at the MIT. ConceptNet includes information extracted from data sources such as the Wikipedia and Wiktionary projects, a subset of DBPedia from the Leipzig University which contains information extracted from the infoboxes on Wikipedia articles and English facts from the word game Verbosity, formerly run by the GWAP project, an academic project at Carnegie Mellon University.

We agree with Baydin et. al [1] regarding noise in ConceptNet. We found biased relations against political subjects, gender issues, sexual or sexist remarks, incomplete or erroneous concepts, funny statements and incorrect relations such as the following: *cell_phone,isa,cat*; *montain_ion,isa,cat*; *food,isa,cat*; *prion,isa,prokaryote*; *woman,purposeof,cook*; etc.

We decided to clean ConceptNet from many of the above issues although many do yet remain. We did this not only for these experiments but expecting the future use of an optimized ConceptNet in further experiments on our research projects. Also removed where ambiguous concepts such as *this*, *that*, pronouns which do not define a clear subject, definite and indefinite articles, lengthy concepts as *thin_material_that_be_fold_entirely_around_object* and

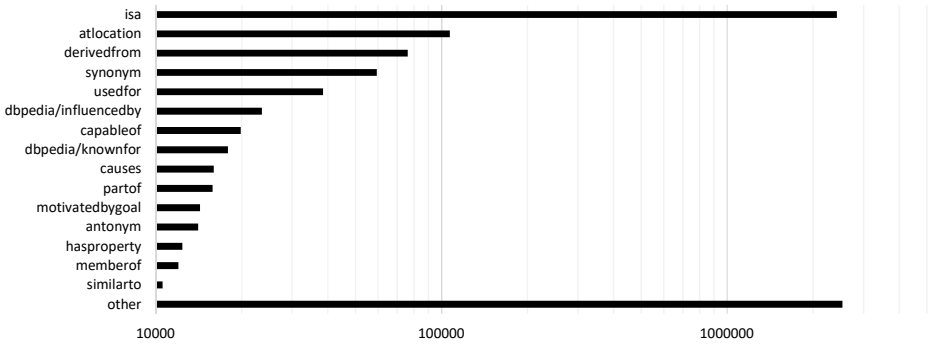


Fig. 2: A portion of the relations with the shown label and their number in the processed ConceptNet semantic graph. Notice the logarithmic scale.

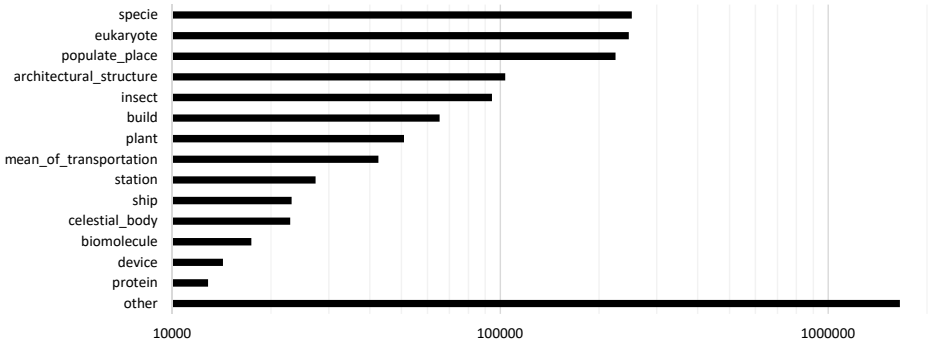


Fig. 3: The quantity of the source concepts of *isa* relations from a portion of the processed ConceptNet semantic graph. Notice the logarithmic scale.

large_bird_whimsically_think_of_as_wear_tuxedoes, etc. Additionally, many incorrect relations were reversed, e.g., *person,isa,athlete* $\rightarrow$ *athlete,isa,person*.

From the pre-processed ConceptNet graph we chose the largest graph component which contains 1229508 concepts and 1791604 edges. To give an idea of the information present in the processed ConceptNet the number of relations of various types are displayed in Fig. 2 and the concepts T which are targets of *isa* relations are shown in Fig. 3.

The input space extracting algorithm, Domain Spotter, was run with a population of 256 chromosomes, a limit of 1024 epochs, binary tournament selection with the strongest chromosome having a winning probability of 85%. The bridge jumping parameter γ was set to 2. We varied the τ parameter and run the Spotter multiple times in order to obtain pairs of input spaces with differing characteristics such as the shown in Table 1. This table is described as follows:

- The first column contains the τ parameter used in the Domain Spotter,
- the **bridge** column contains the required bridging concept which interconnects the two input spaces,

- the **degree** column shows the degree of the bridge node (number of incoming/outgoing relations),
- the **deep** column represents how many relations/edges is the farthest concept of each of the input spaces away from the bridge node,
- $\#s_0$ and $\#s_1$ are respectively the number of concepts present in input spaces 0 and 1,
- the **intersect** column contains the number of concepts intersecting both input spaces (and as such present in both of them),
- **concepts** shows the combined amount of concepts in both input spaces and is equal to $\#s_0 + \#s_1 - \#(s_0 \cap s_1) - 1$. The reduction by one is because of the bridge concept being present in each input space (thus twice in the sum of their cardinality) but only once in their intersection.

After obtaining the input space pairs using the Domain Spotter, we gave each pair to the three mappers (optimum, jMapper and EEmapper), executed each one seven times, recording the executing time and the resulting output mappings. The source code of the Optimum mapper and jMapper where changed to add a time-out and to return their best result until then, instead of using their original implementation of waiting forever and returning a result. This time-out and EEmapper's running time was limited to 3600 seconds.

Table 2 contains the cardinality (number of concept pairs) of the mappings obtained from the three mappers, including their standard deviation. Table 3 shows the time (in seconds) the optimum mapper and jMapper took to generate a mapping, except in the case of EEmapper this represents the average time the mapper needed to generate a mapping as big (in number of concept pairs) as the best mapping extracted by one of the other mappers. This is because EEmapper only stops its execution either when it reaches a limit of epochs or a time limit.

The two figures (Figs. 4a and 4b) compare the real-time cardinality of the mappings generated by all the mappers during their execution. Because we can not output both the optimum mapper and jMapper mappings in real-time, we terminate them after a time-out of 3600 seconds (one hour). Therefore their graphs are shown as a linearly interpolated lines between their start at $t = 0$ seconds and their termination at either $t = 3600$ seconds or when they complete.

All the experiments and mappers were executed on a Intel Xeon X3470 with 32 GB RAM, Windows 10 x64, Java JDK 9.0.4 and with JVM settings -Xms8g -Xmx24g. The EEmapper ran with 4 parallel threads to minimize cache inefficiency (pollution and misses) and the memory bottleneck, given the dispersive nature of the semantic graphs in computer memory [16,23].

5 Discussion

The proposed EEmapper extracted the largest mappings for 5/8 of the experiments. Other than the experiments: $\tau = 0.1 \wedge \text{bridge} = \text{hiram\_ohio}$, $\tau = 0.25 \wedge \text{bridge} = \text{vascularity}$, $\tau = 0.25 \wedge \text{bridge} = \text{venography}$, the mapper started outputting large mappings after thirty seconds of execution or less (Figs.

Table 1: Input spaces extracted by the Domain Spotter from the processed ConceptNet knowledge base.

τ	**bridge**	**degree**	**deep**	**#s0**	**#s1**	**#intersect**	**concepts**
0.00	exercise_physiology	2	4	4841	124	0	4964
0.01	redwatch	2	3	13	17	0	29
0.01	venography	2	4	42	21	0	62
0.10	hiram_ohio	2	2	3	202368	0	202370
0.10	horror_fiction	2	4	17919	19742	1594	36066
0.25	redwatch	2	5	3379	40808	220	43966
0.25	vascularity	2	4	412593	249954	8682	653864
0.25	venography	2	6	52454	4349	701	56101

Table 2: The cardinality of the mappings for each experiment and mapper. The values shown in **bold** represent the largest mappings.

τ	**bridge**	**optimum**	**jMapper**	**EEmapper**
0.00	exercise_physiology	14 ± 0	1 ± 0	**28.333 ± 1.506**
0.01	redwatch	**10** ± 0	0 ± 0	6.000 ± 0.000
0.01	venography	**11** ± 0	0 ± 0	6.000 ± 0.000
0.10	hiram_ohio	2 ± 0	0 ± 0	**5.833 ± 1.329**
0.10	horror_fiction	13 ± 0	**3864** ± 0	1175.833 ± 10.722
0.25	redwatch	12 ± 0	398 ± 0	**1410.400 ± 8.877**
0.25	vascularity	2 ± 0	0 ± 0	**7318.600 ± 35.529**
0.25	venography	18 ± 0	1174 ± 0	**2277.600 ± 8.019**

4a and 4b). In the last two mentioned experiments, including the case $\tau = 0.25 \wedge \text{bridge} = \text{redwatch}$, the EEmapper extracted from the input spaces mappings with far higher number of concept pairs when compared with the other mappers and in less than 1/6 of their time. An example of a considerable large mapping is shown in Fig. 5.

In the experiment $\tau = 0.01 \wedge \text{bridge} = \text{redwatch}$, the optimum mapper was able to generate the largest mapping with EEmapper reaching 60% of the optimum mapping. In our opinion, this experiment matches the limit where the complexity of the problem at hand is simple enough that a stochastic solver is not justified for two reasons: 1) an exhaustive solver finds the optimum answer faster than a stochastic algorithm and 2) these situations are straightforward for the optimum mapper but yet complex enough that the probabilistic nature of the EEmapper reduces its likelihood of obtaining the perfect answer.

jMapper was unsuccessful in the second to fourth experiments shown in Table 2, although it managed to extract one pair of concepts from the input spaces in the experiment $\tau = 0.0 \wedge \text{bridge} = \text{exercise\_physiology}$. This is due to the minimal intersection between the two input spaces (Table 1) with this intersection comprised of only the bridge concept. This minimal intersection restricts the usage of the Triangle rule, fundamental to Sapper and thus jMapper. This de-

Table 3: Time (in seconds) taken by each mapper in each experiment. The times in **bold** represent the time required to obtain the largest mapping.

τ	**bridge**	**optimum**	**jMapper**	**EEmapper**
0.00	exercise_physiology	3644.666 ± 85.561	0.875 ± 0.247	**13.793 ± 0.881**
0.01	redwatch	**0.342 ± 0.147**	0.000 ± 0.000	3600.078 ± 0.016
0.01	venography	**3602.435 ± 7.588**	0.000 ± 0.000	33.037 ± 0.016
0.10	hiram_ohio	0.235 ± 0.181	0.379 ± 0.008	**207.844 ± 15.870**
0.10	horror_fiction	3631.042 ± 54.753	**1836.532 ± 63.679**	3602.374 ± 1.277
0.25	redwatch	3651.080 ± 52.964	228.895 ± 0.812	**34.667 ± 1.562**
0.25	vascularity	3633.345 ± 91.153	3612.787 ± 32.445	**425.866 ± 54.790**
0.25	venography	3615.587 ± 17.392	837.980 ± 12.292	**56.602 ± 2.130**

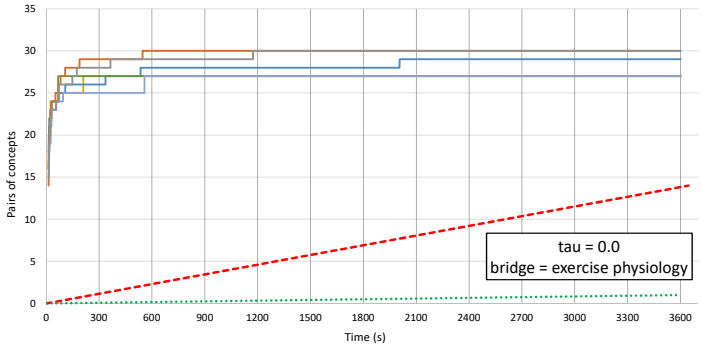
monstrates that jMapper has difficulties finding analogies when the input spaces have little to no concepts in common.

In experiment $\tau = 0.1 \wedge$ bridge = horror_fiction jMapper managed to extract the largest mapping. Our hypothesis is that this experiment contains a large amount of concepts in the intersection of the two input spaces, favourable to Sapper's Triangle rule. This allowed jMapper to further trigger the Square rule and easily expand its mappings. We wonder if, were it given more execution time and computational resources, jMapper would also have extracted a large mapping in the experiment $\tau = 0.25 \wedge$ bridge = vascularity.

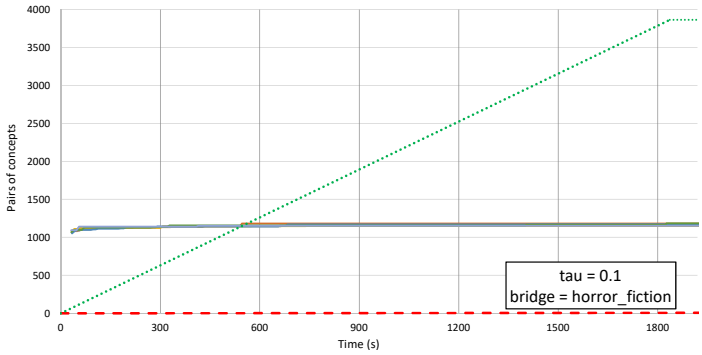
We find important to mention that the mapper is currently a stochastic mechanism without intricate mutation and crossover mechanisms typically expected from evolutionary algorithms and yet it obtained larger results when compared with the other mappers. This emphasises the complexity of the task at hand - extracting analogies from large semantic graphs - and the importance of not obtaining perfect results, but mappings "good enough" for their purpose. This is observed in 5/8 of the experiments except the three following: $\tau = 0.01 \wedge$ bridge = redwatch, $\tau = 0.01 \wedge$ bridge = venography, $\tau = 0.1 \wedge$ bridge = horror_fiction.

Lastly, given the main purpose of the proposed EEmapper - to be used in CB experiments - we find crucial that a first inspection of the mappings' semantic structure is made. Although we expect a deeper examination of this subject in a future paper, we illustrate in Fig. 6 a small region of a mapping generated in experiment $\tau = 0.1 \wedge$ bridge = horror_fiction. Part of the structure of the input spaces is present in the entire mapping, for instance *anemone,atlocation,reef* and *shark,atlocation,reef*.

Noise from ConcepNet is visible in relations such as *shark,isa,asbestos*. However, given this noise and the association of distantly related or even unrelated pairs of concepts in the mapping, we find promising the usage of our EEmapper in the next step of our research - its implementation in a CB framework. For example, who knows if the creative system elaborates a story of a *parrot* named *bug* who works as a *microphone* in the *amazon* rainforest...



(a)



(b)

Fig. 4: Number of concept pairs being extracted during the mappers real-time execution for two experiments. The optimum mapper and jMapper are drawn as straight lines interpolating their starting and ending times. The mappers are represented with the following line strokes: red dashed - optimum mapper; green dotted - jMapper; remaining line strokes - EEmapper.

6 Conclusions and future work

We have proposed a fast multi-threaded mapper for finding analogies in real-time. It was compared against an optimum mapper developed by us and against jMapper, based on Sapper. EEmapper was found to outperform the other mappers in large semantic graphs in the order of at least half a million of interconnected concepts. As far as we know, our mapper is the first of its kind to handle semantic graphs in this level of proportion and to scale beyond. The mapper is based on evolutionary principles and thus straightforward to adapt its fitness function should the mappings have to adhere to different criteria.

We find important to mention that the proposed mapper may appear somewhat elementary but we think that given the purpose of handling large semantic networks and future real-time interaction with a conceptual blending module, as a first step the mapper should be the fastest possible. Hence this requires a

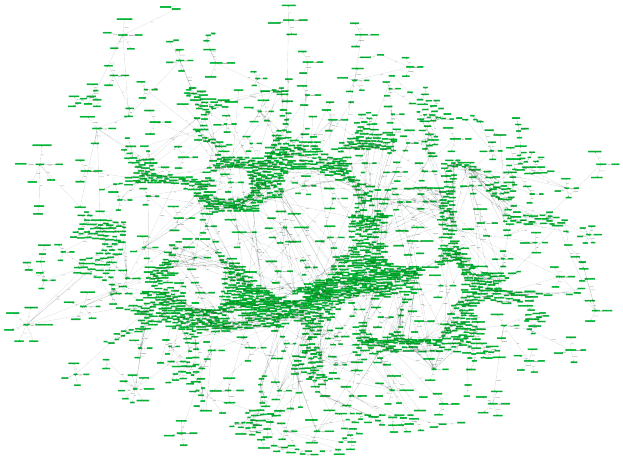


Fig. 5: One of the possible mappings extracted by EEmapper from the experiment $\tau = 0.25 \wedge \text{bridge} = \text{venography}$. It contains roughly two thousand concept pairs. Each vertex in the graph is in the form $(\text{concept}_l, \text{concept}_r)$.

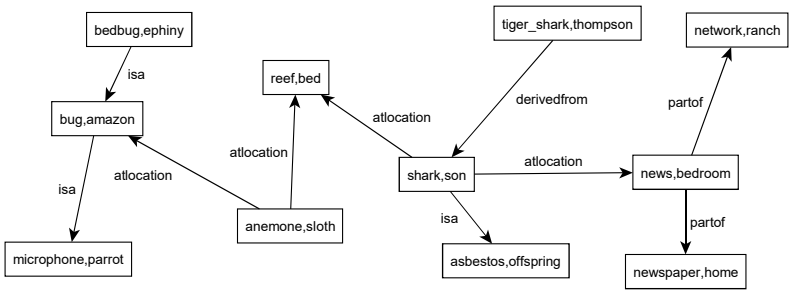


Fig. 6: A small region of a mapping generated by EEmapper in experiment $\tau = 0.1 \wedge \text{bridge} = \text{horror\_fiction}$.

GA with a fast evolution, swift genetic operations and quick fitness evaluations because of existence of a blending module down the pipeline.

In the future we expect further development of our EEmapper in the form of refinements in its evolution algorithms, the addition of the genetic crossover operator and improvements in the mutation of mappings. Moreover, the mappings will undergo a semantic evaluation during their evolution within the EEmapper in order to be consistent with the CC System that we hope to realize in the future.

Acknowledgements. João Gonçalves is funded by Fundação para a Ciência e Tecnologia (FCT), Portugal, under the PhD grant SFRH/BD/133107/2017.

References

1. Atilim Günes Baydin, Ramon López de Mántaras, and Santiago Ontañón. Automated generation of cross-domain analogies via evolutionary computation. *CoRR*, abs/1204.2335, 2012.
2. Margaret Boden. *The Creative Mind: Myths and Mechanisms*. London: Weidenfeld and Nicholson, 1990.
3. João Miguel Cunha, João Gonçalves, Pedro Martins, Penousal Machado, and Amílcar Cardoso. A pig, an angel and a cactus walk into a blender: A descriptive approach to visual blending. In *Proceedings of the Eighth International Conference on Computational Creativity (ICCC 2017)*, 2017.
4. Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989.
5. Gilles Fauconnier and Mark Turner. *The Way We Think*. New York: Basic Books, 2002.
6. R. M. French. The computational modeling of analogy-making. *Trends in Cognitive Sciences*, 6(5):200–205, 2002.
7. Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2), 1983.
8. Dedre Gentner, Keith J. Holyoak, and Boicho N. Kokinov. *The Analogical Mind: Perspectives from Cognitive Science*. Bradford Book, 2001.
9. João Gonçalves, Pedro Martins, and Amílcar Cardoso. Blend city, blendville. In *Proceedings of the Eighth International Conference on Computational Creativity (ICCC 2017)*, 2017.
10. João Gonçalves, Pedro Martins, António Cruz, and Amílcar Cardoso. Seeking divisions of domains on semantic networks by evolutionary bridging. In *The Twenty-Third International Conference on Case-Based Reasoning (ICCBR)*, Frankfurt, Germany, 2015. CEUR, CEUR.
11. Christine Hall and Pat Thomson. Creativity in teaching: what can teachers learn from artists? *Research Papers in Education*, 32(1):106–120, 2017.
12. Raquel Hervás, Rui P. Costa, Hugo Costa, Pablo Gervás, and Francisco C. Pereira. Enrichment of automatically generated texts using metaphor. In *Mexican International Conference on Artificial Intelligence*, pages 944–954. Springer, 2007.
13. Raquel Hervás, Francisco C Pereira, P Gervás, and Amílcar Cardoso. Cross-domain analogy in automated text generation. In *Procs. 3rd Joint Workshop on Computational Creativity, ECAI 2006, Riva del Garda, Italy*, 2006.
14. Douglas R Hofstadter. Analogy as the core of cognition. *The analogical mind: Perspectives from cognitive science*, pages 499–538, 2001.
15. Tobias Kötter, Kilian Thiel, and Michael R Berthold. Domain bridging associations support creativity. *Proceedings of the International Conference on Computational Creativity, pp. 200–204*, 2010.
16. John McCalpin. Memory bandwidth and machine balance in high performance computers. pages 19–25, 12 1995.
17. T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

18. Francisco C. Pereira and Amílcar Cardoso. The horse-bird creature generation experiment. *AISB Journal*, 1(3), 2003.
19. Francisco Câmara Pereira. *Creativity and AI: A Conceptual Blending approach*. PhD thesis, University of Coimbra, Jan 2005.
20. Marco Schorlemmer, Alan Smaill, Kai-Uwe Kühnberger, Oliver Kutz, Simon Colton, Emilios Cambouropoulos, and Alison Pease. Coinvent: Towards a computational concept invention theory. In *Proceedings of the 5th Int. Conference on Computational Creativity, ICCC-14, Ljubljana, Slovenia*, 2014.
21. Robert Speer and Catherine Havasi. Representing general relational knowledge in conceptnet 5. In *LREC*, pages 3679–3686, 2012.
22. Georgi Stojanov and Bipin Indurkhya. Perceptual similarity and analogy in creativity and cognitive development. In *Computational Approaches to Analogical Reasoning: Current Trends*, pages 371–395. Springer, 2014.
23. John R Tramm and Andrew R Siegel. Memory bottlenecks and memory contention in multi-core monte carlo transport codes. *Annals of Nuclear Energy*, 82:195–202, 2015.
24. Tony Veale and Mark Keane. The competence of sub-optimal structure mapping on hard analogies. In *Proceedings of the International Joint Conference on Artificial Intelligence*. IJCAI-97, 1997.
25. Tony Veale, Diarmuid O'Donoghue, and Mark Keane. Computability as a limiting cognitive constraint: Complexity concerns in metaphor comprehension about which cognitive linguists should be aware. *Cultural, Psychological and Typological Issues in Cognitive Linguistics: Selected papers of the bi-annual ICLA meeting in Albuquerque*, pages 129–155, 01 1995.
26. Patrick H. Winston. Learning and reasoning by analogy. *Commun. ACM*, 23(12):689–703, 1980.
27. Martin Žnidaršič, Amílcar Cardoso, Pablo Gervás, Pedro Martins, Raquel Hervás, Ana Oliveira Alves, Hugo Oliveira, Ping Xiao, Simo Linkola, Hannu Toivonen, Janez Kranjc, and Nada Lavrač. Computational creativity infrastructure for online software composition: A conceptual blending use case. In *Proceedings of the Seventh International Conference on Computational Creativity (ICCC 2016)*, Paris, France, 2016. Sony CSL, Sony CSL.