ChemGrid: An Open-Ended Benchmark Domain for an Open-Ended Learner

Miklos Kepes
Cross Labs
Cross Compass, Ltd.
Kyoto, Japan
kepes.miklos@gmail.com

Nicholas Guttenberg
Cross Labs
Cross Compass, Ltd.
Kyoto, Japan
ngutten@gmail.com

L. B. Soros
Cross Labs
Cross Compass, Ltd.
Kyoto, Japan
lisa.soros@cross-compass.com

***Abstract*—This paper introduces ChemGrid, which is a novel video-game-based domain for exploring open-ended artificial intelligence capable of setting arbitrary goals for itself. In this domain, agents must satisfy a minimal goal of crafting a preset "survival molecule" but otherwise can pursue their own agendas related to crafting in an artificial chemistry based on the concept of pathway complexity from artificial life. In particular, agents can discover new molecular recipes by joining existing recipes together and breaking them apart. This paper explores the design space afforded by the artificial chemistry. The main contribution is the introduction of ChemGrid itself, which is intended as a future benchmark domain for general game-playing algorithms.**

***Index Terms*—Game competitions, benchmarks, multi-agent systems, open-endedness, artificial chemistry**

I. Introduction

Open-ended algorithms continue to produce novel and interesting outcomes even when run forever. Such algorithms are particularly important in the context of two distinct yet potentially related fields: 1) machine learning and 2) generative systems and artificial life. Open-ended machine learning systems, if they exist, would be a major accomplishment because they would be capable of learning to solve arbitrary problems and not just specific tasks. Open-ended generative systems (such as evolution on Earth and in artificial life worlds) are interesting in their own right, but also because they might provide a route to open-ended machine intelligence [1].

Games are an important and underexplored domain for open-endedness research, and for AI research in general, because they abstract key features of real world tasks that require general intelligence into a more tractable format. Games thus far have proven useful domains for testing algorithms for specific components of artificial intelligence such as planning and visual processing, with different game genres providing different kinds of challenges for both human and machine intelligence. Many current AI benchmark domains are inherently limited because they only measure an agent's ability to complete a narrow, pre-defined task. However, achieving open-ended intelligence first requires the existence of domains that enable a diversity of nontrivial agent behaviors.

This current work introduces a radically new game-based environment for exploring open-endedness in the context of multi-agent systems. On an individual level, agents must learn to craft a "survival molecule" in a simple artificial chemistry, but otherwise are free to pursue their own molecule crafting goals. Importantly, there is no inherent reward signal other than a binary indication of whether or not a target "survival molecule" has been built (providing a natural minimal criterion for success). Agents can also participate in a contract-making system (described later) whereby they can offer rewards to each other for achieving agent-defined crafting goals.

The new ChemGrid environment is meant to serve as a new benchmark for both the artificial life and game AI communities. This introductory paper explores in depth the properties of the artificial chemistry system underlying the environment's crafting system and contract-based economy.

II. Related Work

Games have long served as benchmarks for artificial intelligence. In particular, there exist some benchmark *collections* that offer diverse sets of independent tasks with the hope that a generally intelligent player could solve them all. Examples include the General Video Game AI (GVGAI) Competition [2] and the Arcade Learning Environment (ALE) [3].

Still, the games contained in these frameworks are not themselves generally open-ended as the environment provides the agent with reward-specified goals such as killing monsters or navigating mazes. In contrast, sandbox environments such as the game Minecraft contain many conceivable tasks within a single environment, with the environment providing little to no incentive for doing any particular thing besides surviving. This approach is similar to the challenge solved by life on Earth, but is implemented in relatively few game benchmarks. One notable exception is the Minecraft-inspired Voxelbuild [4], wherein evolved agents placed blocks on a plane. The recent EvoCraft framework and competition on open-endedness [5] was additionally inspired by Minecraft, noting that human players have proven capable of building complex artifacts such as computers in the game and asking whether sufficiently open-ended algorithms could achieve similar feats. Of course, Minecraft is not the only possible domain for experiments in open-ended creativity. The SimSim framework [6], inspired by The Sims, focuses on interior design in the context of minimal criteria for viability; any design that meets all of a simulated agent's basic needs is considered viable, and all others are not.

III. Background

This section reviews concepts from artificial life that conceptually ground the work presented in this paper.

A. Minimal criterion evolutionary search

The minimal criterion paradigm frames evolution as survival of the *fit enough* instead of survival of the *fittest*. Prior work on the artificial life world Chromaria [7] showed a binary minimal criterion for viability (as opposed to gradient-based fitness) to be a key prerequisite for for open-ended evolution; without such a mechanism, evolution stagnated. This idea was later adapted into more applied evolutionary algorithms, including Minimal Criterion Coevolution [8] and the Paired Open-Ended Trailblazer (POET) [9]. Another algorithm related to minimal criterion search is the Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm [10], which maintains and evolves a population of individuals that satisfy a binary viability criterion alongside a population that doesn't.

B. Pathway Complexity

Pathway Complexity [11] is a complexity measure for arbitrary objects. Essentially, this measure quantifies the shortest path from basic building blocks to a complex structure as the number of join operations required to compose an object. The original authors give the example of building the string "banana" from individual letters and already-built combinations of letters. By building "ba" (b + a), then "na" (n + a), then "nana" (na + na), then finally "banana" (ba + nana), we get a final pathway complexity of 4. Pathway Complexity can be applied whenever an object is composed of discrete building blocks, which will be true of molecules built by players in ChemGrid. It is preferable to measures such as Shannon entropy and Kolmogorov complexity because it is not maximized by randomness and instead in theory identifies structures with interesting or lifelike complexity.

C. Novelties and "door-opening" states

Banzhaf et al. [12] identify three kinds of novelties that may occur in evolving systems, with the ongoing production of novelty being required for open-endedness in evolution and artificial life. Taylor [13] connects these types of novelties with Boden's kinds of creativity, and additionally notes that "...kinds of open-endedness involving expansive and transformational novelties...both involve the discovery of door-opening states in P-space that open up an expanded space of new adjacencies". The artificial chemistry in ChemGrid was designed such that simulated molecules, once constructed by agents, open doors to constructing other molecules in a theoretically infinite network.

IV. ChemGrid Framework

A screenshot of the main ChemGrid interface is shown in Figure 1. The window is 256x256 pixels and is intentionally kept small so that raw pixels can be input to a neural network. This way, the agent must learn to play the game without being given domain-specific knowledge and the hope is that gameplaying abilities will generalize to other games.

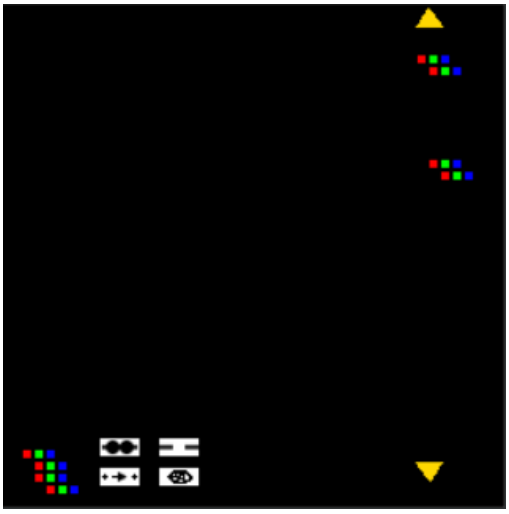


Fig. 1: **Main ChemGrid interface.** The bottom left corner depicts a "survival molecule" that serves as a minimal criterion for successful agent play. To the right are buttons representing, clockwise from top left, *join*, *break*, *view available contracts*, and *set contract* operations. The column on the right contains the agent's (scrollable) inventory, which represents all molecules the agent has ever constructed.

At each time step, a single player instance can click once anywhere on the screen. The basic actions available to the agent, assuming it can navigate the interface successfully, are:

- **Join** (Figure 2) : The agent selects two molecules from its inventory and connects them to form one combined molecule. A join operation is valid if the components are placed adjacent to one another, i.e. they form a single connected component. This operation exists so that atomic complexity can be measured as pathway complexity.
- **Break** (Figure 3): The agent selects one molecule from its inventory and specifies a bond to remove that will break the molecule into two disjoint sets of atoms (i.e. two new molecules). Along with the join operations, this operation also exists so that atomic complexity can be measured as pathway complexity.
- **Set a contract**: The agent selects one molecule from its inventory as a "payout" that another agent will receive if it constructs a target molecule (distinct from the global survival molecule) specified by the player. Nothing happens if a contract goes unfulfilled, so there is no cost to creating contracts. The contract system provides a mechanism whereby agents can communicate their goals to each other and collaboratively participate in open-ended exploration of the design space.
- **View contracts**: The agent can view all contracts (payout/target molecule pairs) that have been submitted by other agents.

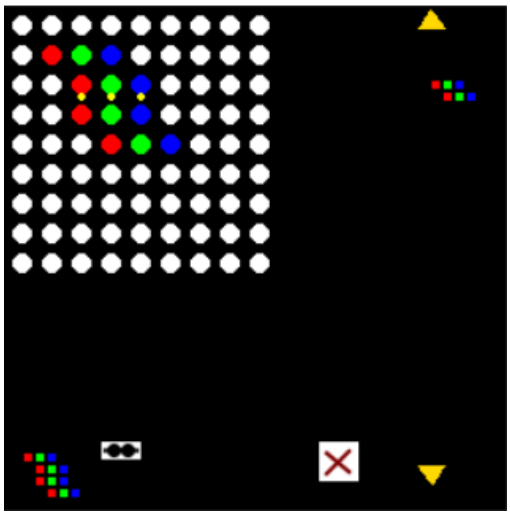


Fig. 2: **Join interface.** A molecule is selected from the inventory and then placed on a grid of atoms. A second molecule is selected from the inventory and can be placed anywhere on the grid. If placing the two molecules adjacent to each other would result in creating one contiguous molecule, possible new bonds appear in yellow. After construction, the new molecule will appear in the agent's inventory.

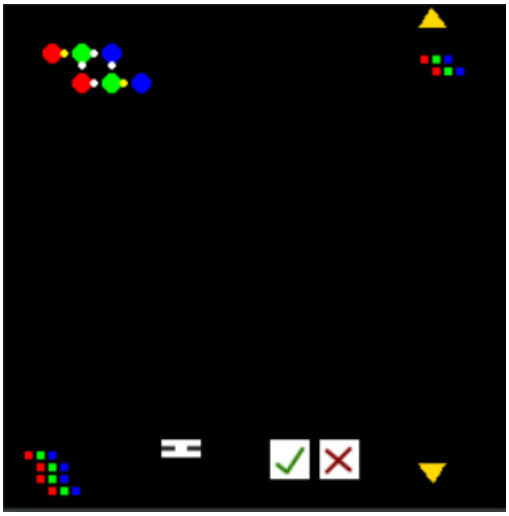


Fig. 3: **Break interface.** The player has selected a molecule (shown alone at the top of the inventory column on the right) and must now select a valid break point. A bond is a valid break point if removing the bond will result in two disjoint sets of atoms. Valid break points are shown in yellow and invalid break points are shown in white. The two new submolecules will then appear in the agent's inventory.

Source molecules remain in the inventory after join and break operations. As such, the inventory is a record of every "recipe" that an agent has discovered rather than a set of items in the agent's possession. Inventory size is currently unlimited, which may change in future ChemGrid versions if necessary.

V. Design Space Exploration

The rest of this paper explores the space of molecules possible in ChemGrid, the role of the join and break rules, and the contract system in the building process. The goal is to show why this environment has the potential to produce interesting dynamics and properties, including open-ended characteristics.

Because developing agents that can play the game end-to-end through its interface is an open challenge and the ultimate goal behind the creation of the ChemGrid framework, and furthermore because the aim in this section is to give an insight into the molecule space, agents herein can access the inventories, targets (survival molecules) and contracts and use the join/break/contract actions directly. Throughout these explorations, the grid size, the number of atom types, the initial inventories, and the target molecules are all also varied.

An interesting question is whether agents are better off using contracts and thereby acquiring mutually beneficial molecules if the means are given. That is, can contracts reduce the number of steps it takes to find targets as the number of agents increases? This question can be generalized: are there molecules that are generally worth exchanging? Are there building blocks that are more valuable than others, i.e. modules that greatly simplify otherwise challenging tasks? These "hub" molecules would be "door-opening" gateways to new opportunities. They are expected to be a building block towards many different targets and also relatively difficult to construct, therefore exchanging them via contracts is more economical than building them from scratch multiple times independently. Note that it is possible to reach any target from the starting inventory using only joins and breaks, but that utilizing contracts can provide a more efficient route to targets.

A. Space of possible molecules

The power of human language to efficiently express practically any thought or concept comes from its combinatorial structure [14]; letters, utterances, and sentences can be combined to construct new, more complex language building blocks. Similarly, agents in ChemGrid can ever-combine molecules to construct potentially more complex and delicate ones. Due to this combinatorial property, the number of possible molecules grows exponentially with the number of molecules used to build them. Figure 4 shows the number of valid molecules by the total number of atoms, when using three different atom types on an 6x6 grid for total number of atoms 1-7. Figure 5 shows a sample of randomly-generated molecules following a repeated application of the join and break rules on randomly-selected molecules. This sample was obtained from the resulting $\sim 400,000$ items. This example gives an idea about the number of opportunities for agents in this environment. However, not all of these opportunities have equal significance; it is expected that some of the structures will be more complex (e.g. according to the pathway complexity) or valuable (e.g. according to their scarcity and reusability [Section V-D]).

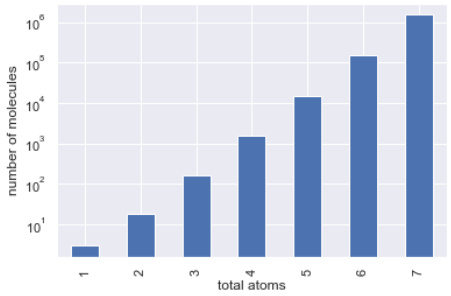


Fig. 4: **Number of molecules by total number of atoms.** The number of possible molecules grows exponentially with the number of atoms contained in the molecules.

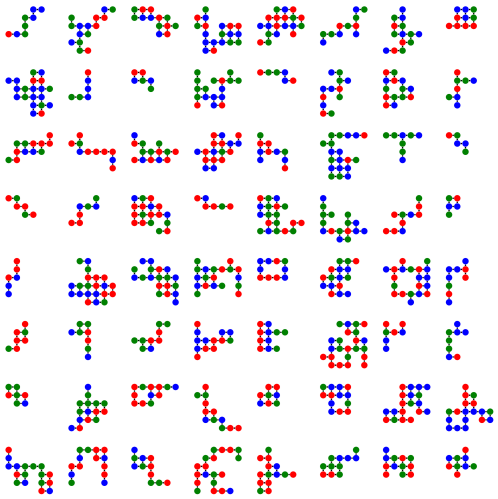


Fig. 5: **Sample of randomly generated molecules on a 6x6 grid.** Obtained by following a repeated application of the join and break rules on randomly sampled molecules from a growing archive. The final sample was generated from the resulting $\sim 400,000$ items.

B. Molecule-building agents

For the purposes of exploring the molecule building space, we start with an approach that uses a hard-coded agent logic. Also, normally, for an agent to be able to play ChemGrid it needs to process RGB images of the screen, and come up with an action in the format of the coordinates of the next mouse click on the game interface, as shown in Figure 6. Furthermore, while completing target molecules are not intended to be the ultimate objectives of ChemGrid agents (rather, they should be viewed as minimal criteria for survival), in these experiments we build agents whose goal is completing their targets so that we gain insight into how the ChemGrid rules influence the molecule building trajectories and the usage of contracts.

The backend of ChemGrid is converted into an OpenAI Gym [15] like environment[1], with which agents can iteratively

[1]https://github.com/mekhlos/chemgrid

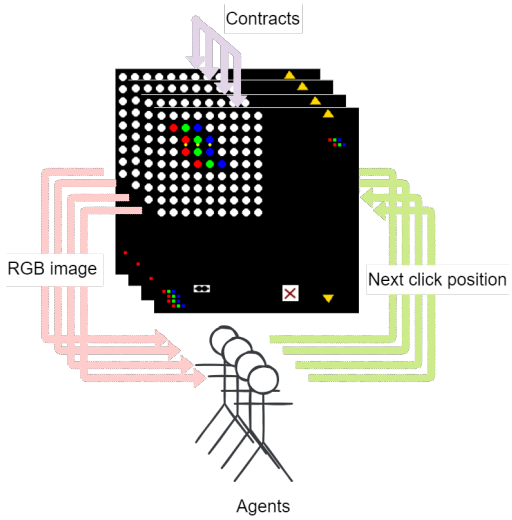


Fig. 6: **Agent-game interaction.** Agents receive RGB screen images as states and send mouse click positions as actions. Each agent has its own interface and can only interact with other agents by creating, viewing, and completing contracts.

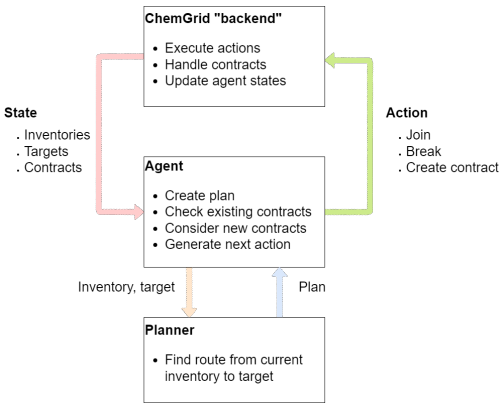


Fig. 7: **Agent environment interaction using ChemGrid backend.** Agents receive all the agents' state from the environment, containing inventories, targets, and the current contracts, then they can create a trajectory plan from its current inventory to its target using join and break and use this plan to generate the next action. After considering other agents' plans, an agent can also create contracts that would shorten the trajectory of both agents or it can evaluate and work towards existing contracts.

interact through time, as in Figure 7. In this setup, n agents are initialized with initial inventories and targets. At each step, each agent can generate an action to either join two molecules, break a molecule, or generate a contract promising a molecule upon completion of a requested molecule, as described in Section IV. The selected actions are then executed in the environment and the updated inventories and contracts are returned together with a "done" indicator for each agent indicating which agents have reached their targets.

At every step, agents

- check if there are existing contracts worth pursuing. If yes, they generate a plan from current inventory to the chosen sub-target using the planner module detailed below, and return the first step towards the sub-goal.
- otherwise they check if other agents currently possess or are likely to possess molecules that would reduce the steps to reach their targets based on their plans. If they can offer something from their own present or future inventories in exchange for such desirable items they generate a new contract. Contracts are supposed to help both participants reach their targets faster than separately.
- otherwise, they generate a plan from the current state to their target using the planner module.

Planner: The planner finds a trajectory from a given inventory to the target using only breaks and joins, via

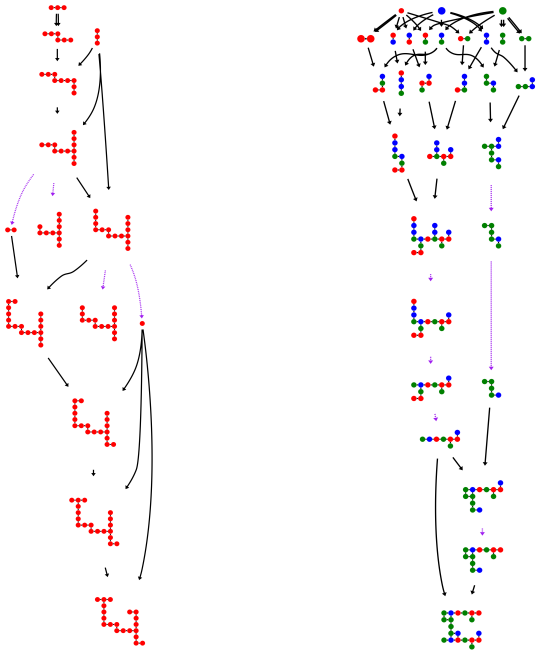
- **Exhaustive search:** building all possible combinations starting from the initial inventory, until it comes across the target. This approach, while guaranteed to eventually reach the target (if possible), is very inefficient, and only applicable for smaller grids, and simpler targets.
- **Heuristic search:** use heuristics such as a similarity measure between molecules and the target to guide the search through the molecule selection process, (e.g. Equation 2). Search is faster, but biases can be introduced, and incremental trajectories such as in Figure 8a can result.

There could be many ways to improve the planning algorithm, including machine-learning-based approaches to predict which molecules to select for join and break. Figure 8a shows an examples of a single target-building agent in action using the heuristic-based planner. As the initial inventory doesn't contain the generally applicable 1x1 and 1x2 molecules, they are obtained through breaks. One disadvantage of this heuristic approach is that the target molecule is built in an incremental manner, i.e. the agent maintains one main working-molecule which it amends incrementally. The number of steps is probably far from the optimal, shortest path. This can be contrasted to the trajectory in Figure 8b, which was obtained from a randomly-built molecule.

C. Experimenting with the rules

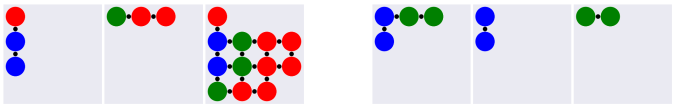
In an alternative approach to joins and breaks, we updated the rules so that to join two molecules one has to specify the type of bond on which to join them, that is a pair of colors representing two (not necessarily) different atom types, see Figure 9a. Then all possible valid combinations are generated where these two types of atoms are adjacent. Breaks are also done by specifying a bond and all atom pairs with this bond are separated forming new molecules (Figure 9b). Only molecules with more than one atom are considered valid, and an operation is only valid if all new items are valid.

Figure 10 shows a sample of randomly-generated molecules following these new rules, while Figure 11 shows an example trajectory for building a molecule with this approach. We see more reuse of molecules compared to the original joins and



(a) Heuristic-based trajectory. (b) Sampled trajectory.

Fig. 8: **Target-building trajectories.** The molecules without incoming arrows correspond to the initial inventory of the agent, while the bottom molecule is the target. Black arrows represent join and purple dashed arrows represent break operations. For the left figure, we used a heuristic-based similarity measure to guide the agent's search towards its target. On the right figure we show a trajectory that was sampled from randomly built molecules and unlike the other trajectory, we see examples of multiple components being built separately then joined, instead of progressively developing a single one.



(a) Join. (b) Break.

Fig. 9: **New join and break rules.** We updated the rules so that to join two molecules one has to specify the type of bond on which to join them and all possible valid combinations are generated where these two types of atoms are adjacent. Breaks are also done by specifying a bond and all atom pairs with this bond are separated forming fragments. On the left figure the first two molecules are joined on all blue-green bonds to from the third. On the right, the first molecule is broken on all green-blue bonds to form the second and third molecules.

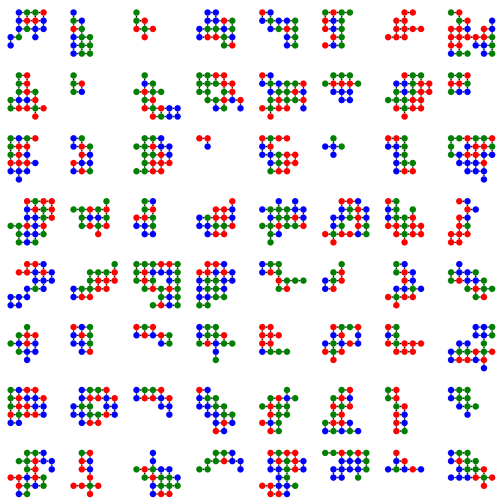


Fig. 10: **Sample of randomly generated molecules when join and break rules are changed.** We obtained this sample from $\sim$ 10000 molecules generated by a repeated application of join and break on a growing archive. Compared to Figure 5, we see, more characteristic, denser molecules.

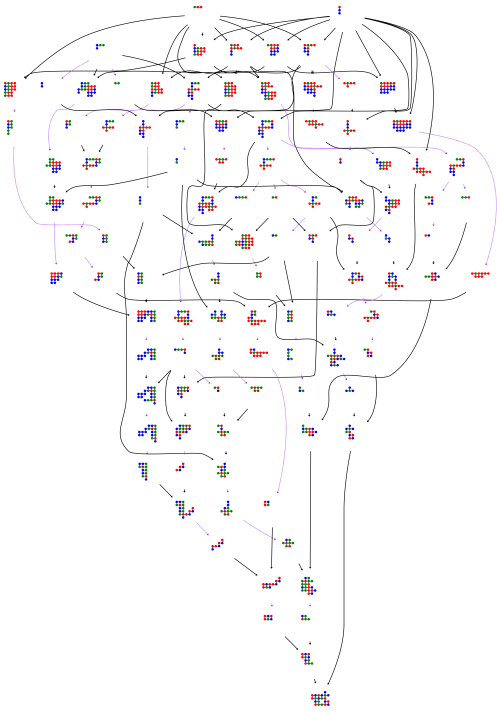


Fig. 11: **Trajectory sample for updated join and break rules.** With the changed rules we see more reuse of molecules compared to the original join and break actions. The space of possible molecules is more constrained and there are less valid combinations, which also makes the emergence of hard-to-come-by, more valuable molecules more likely.

breaks. The space of possible molecules is more constrained and there are fewer valid combinations, which also makes the emergence of rarer, more valuable molecules more likely.

The previous examples show trajectories of individual agents without using ChemGrid's contract mechanism. In the next section we describe contracts more in detail and show some trajectories that involve contracts.

D. The role of contracts

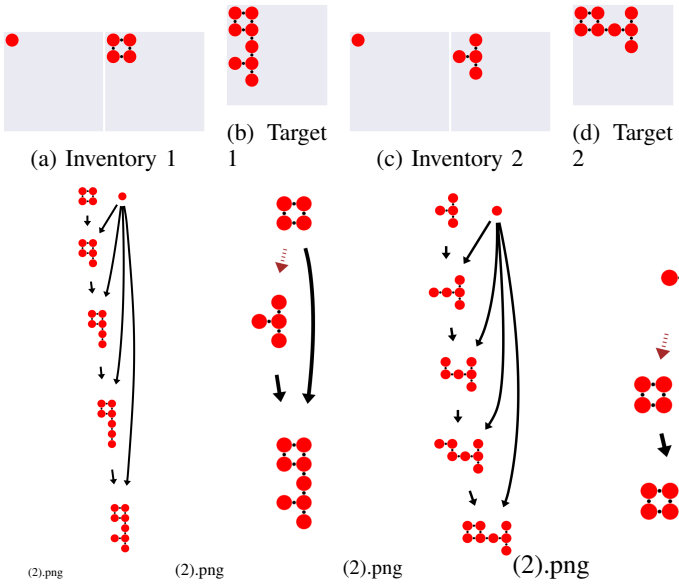
The basic idea behind the contract mechanism is that agents can promise to deliver a certain molecule if another agent completes a requested molecule for them. Agents are obliged to fulfil their promises and deliver if they request was satisfied. It is also possible to promise and request molecules that are not yet built at the time of the creation of the contract. The exchange happens automatically once the requesting agent has the promised molecule and any other agent possesses the requested one. We expect the main role of contracts to be incentives to build new molecules, and in that case agents can decide whether building a requested item is worth it for them. This way agents can create subgoals for each other.

As reasoning about multi-agent environments is not well-explored and -understood it is often desirable to place multi-agent problems into a better-understood single-agent framework. The main challenge in training agents in multi-agent systems is that the system is non-stationary; agents adjust their behaviour as a function of other agents' behaviour, which in turn changes how other agents behave, thus generating a shooting-at-a-moving-target problem. We hope that contracts can somewhat alleviate this issue by giving the agents the ability to tell others what they want instead of needing them to guess and have model their motivations.

ChemGrid is a multi-agent environment where interactions between agents are only possible via contracts. Without contracts, players can only rely on their own inventories for constructing new molecules. In this case, building the target molecule for n agents would take n times as many steps on average as building the target for a single agent.

We set up a two-agent experiment where the benefit of contract usage is clearly shown. In Figures 12a, 12c, 12b, 12d we show the inventories and the targets of the two agents, while 12f and 12h show their trajectories. The targets are composed of the same pair of molecules but have different relative positions to each other. At start, *Agent 1* owns the first component, while *Agent 2* owns the second. Without contracts they would both need to build the missing parts to complete their targets, however, contracts allow them to simply exchange these components and reach their targets faster.

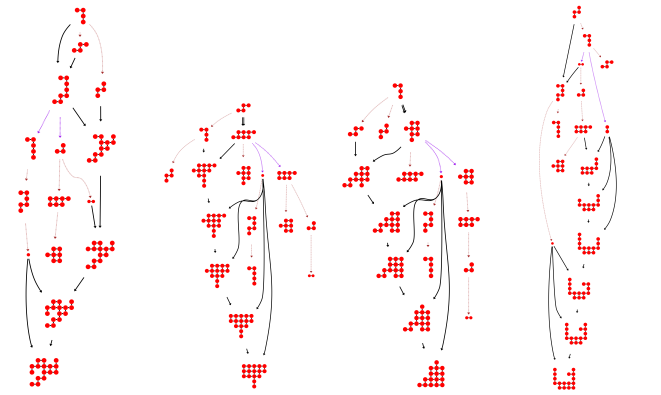
The previous experiment was specifically designed to be ideal for contracts, but we were also curious to see how often contracts emerge automatically if we choose targets randomly. To answer this question, we ran experiments increasing the number of agents and comparing contract usage. As Figure 14 shows, contracts are more likely to emerge as we increase the number of agents; with 32 agents we see about 0.39 contract requests per agent, which adds up to 12.5 total contract



(a) Inventory 1 (b) Target 1 (c) Inventory 2 (d) Target 2

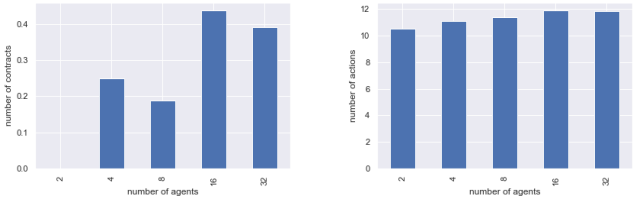
(e) Agent 1 no contract (f) Agent 1 contract (g) Agent 2 no contract (h) Agent 2 contract

Fig. 12: **Benefit of contracts.** Agents reach their targets in two steps instead of four when using them. Here, the targets (fig. (b) and (d)) are composed of the same pair of molecules but have different relative positions to each other. At start, agent 1 owns the first component (fig. (a)), while agent 2 owns the second (fig. (c)). Without contracts they would both need to build the missing parts to complete their targets (fig. (e) and (g)), however, contracts allow them to simply exchange these components and reach their targets faster (fig. (f) and (h)).



(a) Trajectory 1 (b) Trajectory 2 (c) Trajectory 3 (d) Trajectory 4

Fig. 13: **Contract usage emerges when we use non-trivial initial molecules.** We ran four heuristic-based target building agents with random initial inventories and targets. With multi-atom initial inventory molecules, agents are more inclined to utilise contracts (represented by brown dotted edges).



(a) Number of contracts per agent. (b) Number of total actions.

Fig. 14: **Contract utilization vs. number of agents.** With more agents we see more contract usage, however the number of steps to complete the target doesn't decrease. With 32 agents we see about 0.39 contract request per agent (figure (a)) which adds up to 12.5 total contract requests. However, this doesn't seem to decrease the number of steps it took for the agents to reach their targets (figure (b)). This suggests that the agents' contract mechanism needs to be optimized which is subject of our future research.

requests. Note that the experiment was repeated three times with different targets, hence the non-integer result. However, as we can see on the right hand side of the figure, this doesn't seem to decrease the number of steps it took for the agents to reach their targets. Perhaps the molecules acquired through these contracts weren't crucial in building the target molecules. Figure 13 depicts the trajectories of a four-agent example of the target-building experiment. Once again, we see agents use contracts during their trajectories towards their targets.

These experiments show the potential benefit of cooperation through contracts, however, finding the right algorithms and quantifying this benefit is an open challenge in ChemGrid; while an optimal agent would recognize that setting contracts results only in benefits to itself and would take advantage of this fact, preliminary experiments did not yield agents with such behavior.

E. Estimating the value of molecules

When generating contracts in Section V-B, agents estimate how many steps they would save by having a desired molecule in their inventory and compare this to the number of steps it would take to offer something that would save steps for another agent. Based on this we could define a function for molecule value v as

$$v(m, I_t^k, g^k, o) = d(I_t^k, g^k) - (d(I_t^k \cup \{m\}, g^k) + d(I_t^k, o)) \quad (1)$$

where m is the candidate molecule, o is the molecule to offered per the contract, I_t^k is the inventory of agent k at time t, g^k is the target of agent k and d is a function that computes the number of steps needed to build g^k starting from I_t^k. Currently d is the length of the trajectory the planner creates (which currently ignores contracts from the plan).

From another point of view, when selecting breaks and joins (and contracts too), we are curious about how good it is to select certain molecules as the subject of the next action. In this case, the value could represent the negative total number of steps it would take to reach the target if the given molecule was selected for the next action, instead of the number of steps it would save (since it is already part of our inventory).

A molecule that can be used to directly build the target in one step would have a value of -1, if it would take two steps, the value would be -2, etc. This is related to the definition of Q-values in reinforcement learning [16]; Q-values represent the total expected reward in a given state, following a given action. We adopt a version of this approach in our heuristic-based algorithm, approximating the molecule value by its similarity to the target state described by the sim function.

$$v_{\text{mol}}(m, I_t^k, g^k) = \text{sim}(m, g^k) = \frac{\left|m \cap g^k\right|}{|m|} + \alpha \frac{\left|m \cap g^k\right|}{|g^k|} \quad (2)$$

where $|.|$ represents the number of atoms in a given molecule. That is, we take the weighted sum of the portion of the molecule that overlaps with the target and the portion of the target that overlaps with the molecule. Then we use this value in the selection of the next molecule to join or break.

Another interpretation for molecule values can be related to the scarcity/reusability property of molecules from Section V-D. This metric doesn't depend on the current state of the agents, but rather is determined by the join and break rules and other properties of the molecule. Finding a suitable definition for this value is an open question and subject of future research.

VI. Discussion

This paper introduced ChemGrid, a novel video game domain for open-ended task exploration. While investigations focused on open-endedness of the molecule space using the ChemGrid backend, we also experimented with agents that play ChemGrid through its interface. For this purpose, we wrapped the game in an OpenAI Gym environment where states correspond to the visible game screen and actions correspond to mouse clicks. While it is common to use such pixel-based state-spaces (see The Arcade Learning Environment [3]), it is less common[2] to use an action space of the same size (256x256 in our case), which poses an unmet challenge.

It may turn out to be the case that open-endedness in this domain, and in games in general, is a property not just of the game/environment (though open-ended properties of the game are a prerequisite for open-ended gameplay) but of the coupled game-player system. What makes sandbox games so open-ended is that players create their own goals in an otherwise blank canvas environment. It remains an open question how best to motivate Chemgrid agents to set goals for themselves beyond crafting a pre-specified survival molecule, and how to make agents care about the success or failure of other agents (which would prompt the utilization of contracts).

This environment could conceivably become the basis for a formal competition, however the lack of a progressively-increasing score earned by the agent (as in most games used for competitions) makes evaluating agents difficult. One solution might be to use the pathway complexity of all molecules in an agent's inventory after some time as a measure of agent progress, but biasing the agents towards any game-specific goals in particular goes against the open-ended spirit of the game. Another option may be to include human judges as subjective evaluators of constructed molecules, as in the Generative Design for Minecraft Competition [18].

[2]A notable exception is the StarCraft II Learning Environment [17].

Acknowledgement

This work was supported by a grant from GoodAI.

References

[1] K. O. Stanley, J. Lehman, and L. Soros, "Open-endedness: The last grand challenge you've never heard of," December 2017.

[2] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[4] L. B. Soros, J. K. Pugh, and K. O. Stanley, "Voxelbuild: a minecraft-inspired domain for experiments in evolutionary creativity," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 95–96.

[5] D. Grbic, R. B. Palm, E. Najarro, C. Glanois, and S. Risi, "EvoCraft: A New Challenge for Open-Endedness," *arXiv*, Dec 2020. [Online]. Available: https://arxiv.org/abs/2012.04751v1

[6] M. Charity, D. Rajesh, R. Ombok, and L. B. Soros, "Say "sul sul!" to simsim, a sims-inspired platform for sandbox game ai," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 182–188.

[7] L. Soros and K. Stanley, "Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria," in *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press, 2014, pp. 793–800.

[8] J. C. Brant and K. O. Stanley, "Minimal criterion coevolution: a new approach to open-ended search," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 67–74.

[9] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Poet: open-ended coevolution of environments and their optimized solutions," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 142–151.

[10] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, "On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch," *European Journal of Operational Research*, vol. 190, no. 2, pp. 310–327, 2008.

[11] S. M. Marshall, A. R. Murray, and L. Cronin, "A probabilistic framework for identifying biosignatures using pathway complexity," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 375, no. 2109, 2017.

[12] W. Banzhaf, B. Baumgaertner, G. Beslon, R. Doursat, J. A. Foster, B. McMullin, V. V. De Melo, T. Miconi, L. Spector, S. Stepney *et al.*, "Defining and simulating open-ended novelty: requirements, guidelines, and challenges," *Theory in Biosciences*, vol. 135, no. 3, pp. 131–161, 2016.

[13] T. Taylor, "Evolutionary Innovations and Where to Find Them: Routes to Open-Ended Evolution in Natural and Artificial Systems," *Artificial Life*, vol. 25, no. 2, pp. 207–224, 05 2019.

[14] B. de Boer, W. Sandler, and S. Kirby, "New perspectives on duality of patterning: Introduction to the special issue," *Language and Cognition*, vol. 4, no. 4, p. 251–259, 2012.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[17] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[18] C. Salge, M. C. Green, R. Canaan, and J. Togelius, "Generative design in minecraft (gdmc) settlement generation competition," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 2018, pp. 1–10.